Context-Supported Lane Estimation

Understanding the Scene by Learning Spatial Relations Between Semantic Features and Virtual Ground Truth

Von der Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig
zur Erlangung des Grades eines
Doktoringenieurs (Dr.-Ing.)

genehmigte Dissertation

von
Volker Patricio Schomerus
geboren am 08.12.1982
in Osorno

Eingereicht am: 09.08.2016
Disputation am: 27.10.2016
1. Referent: Prof. Dr.-Ing. Friedrich M. Wahl
2. Referent: Prof. Dr.-Ing. Torsten Bertram

2016

Danksagung

Die vorliegende Arbeit ist im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Robotik und Prozessinformatik der Technischen Universität Carolo-Wilhelmina zu Braunschweig entstanden. Für die angenehme Zeit und die technische und freundschaftliche Unterstützung möchte ich allen meinen Kollegen und Freunden am Institut danken. Besonders danke ich Jens Spehr für die wertvollen technischen Diskussionen. Ganz besonders möchte ich Herrn Prof. Dr. Friedrich M. Wahl für seine fachliche Betreuung auch über seine aktive Zeit als Institutsleiter hinaus danken sowie für die Möglichkeit, wissenschaftliche Themen frei und selbstständig bearbeiten zu können.

Während meiner Arbeit hatte ich die uneingeschränkte Unterstützung meiner Familie. Ich möchte meinen Eltern für ihre Geduld und die Ermutigungen bis zur Fertigstellung dieser Arbeit danken. Die Fahrzeugbilder, die für Veranschaulichungen und die Visualisierung von Ergebnissen verwendet wurden, sind Fotos eines Lloyd Alexander TS Modells meines Vaters.

Mein größter Dank gilt meiner Frau Geraldine, die mich bis zum Schluss motiviert und oft bis spät in die Nacht Zeit mit mir am Schreibtisch verbracht hat. Du hast einen großen Anteil an der Fertigstellung dieser Arbeit. Danke!

Braunschweig, August 2016 *Volker Schomerus*

Disclaimer

Die Ergebnisse, Meinungen und Schlüsse dieser Dissertation sind nicht notwendigerweise die der Volkswagen AG.

Zusammenfassung

Fahrerassistenzsysteme und automatisches Fahren gewinnen im Bereich der Mobilität mehr und mehr an Bedeutung. Durch erhöhte Sicherheit und die Möglichkeit einer anderweitigen Nutzung der Reisezeit wird die Entwicklung intelligenter Fahrzeuge die Mobilität der Zukunft neu definieren.
Um die Grenzen der Systeme für vollautomatisches Fahren zu erweitern, versuchen Forschungseinrichtungen und Firmen bereits, neben gut strukturierten Autobahn-Szenarien auch komplexere Fahrzeugumgebungen zu meistern, wie z.B. den Stadtverkehr. Während derzeitige Methoden zur kamerabasierten Fahrzeugumfelderfassung traditionelle Bildsegmentierungs- und Objekterkennungstechniken verwenden, stellt diese Arbeit einen großen Schritt in Richtung umfassenden Szenenverstehens dar. Zu diesem Zweck werden leistungsfähige Methoden des maschinellen Lernens eingesetzt, um die räumlichen Beziehungen zwischen diversen Merkmalen im Kamerabild und der Fahrzeugtrajektorie zu lernen. Das Zusammenführen der räumlichen Beziehungen aller gefundenen Merkmale eines Kamerabildes resultiert in einer sogenannten Verteilungskarte, in die ein Fahrstreifenmodell eingepasst werden kann. Des Weiteren wird mit Hilfe globaler Bildmerkmale der Kontext der aktuellen Szene bestimmt. Es werden verschiedene Möglichkeiten untersucht, mit Hilfe der gewonnenen Kontextinformation die Fahrstreifenerkennung zu verbessern, und es wird erläutert, wie ein solches globales Verfahren mit einer lokalen Methode zur Fahrstreifen- bzw. Fahrstreifenbegrenzungserkennung kombiniert werden kann. Es wird gezeigt, dass viele verschiedene Arten von Merkmalen in der Fahrzeugumgebung wichtige Informationen über die Lage des Fahrstreifens liefern und dass dieser Fahrstreifen im Bild detektiert werden kann, indem dessen räumliche Beziehungen zu diesen Merkmalen modelliert werden. Außerdem wird gezeigt, wie zusätzliches Wissen über den aktuellen Kontext die Qualität der Fahrstreifenerkennung erhöhen kann.

Abstract

Advanced driver assistance systems and automatic driving are becoming more and more important in the market of personal mobility. By increasing traffic safety and allowing the driver to use the traveling time for other activities, the generation of intelligent vehicles creates a new definition of mobility in the future.
To extend the limitations of systems for fully automated driving, research institutions and companies are trying to master more complex vehicle environments like urban traffic. While current approaches for camera-based vehicle environment perception use traditional image segmentation and object detection techniques, this work presents a big step towards comprehensive scene understanding. For this purpose, powerful machine learning methods are applied to learning the spatial relations between several types of features in the camera image and the vehicle trajectory. The registration of these spatial relations for all features in a video frame leads to a distribution map which allows the matching of a lane model. Additionally, the context of the current vehicle environment is determined by extracting global image features.
Several possibilities for improving the lane detection performance with additional context information are analyzed, and the combination of global and local lane or lane border detection methods is proposed. It is shown that many different types of features within the vehicle environment provide important information about the lane and that, by modeling the spatial relations between features and the trajectory of the vehicle, its lane can be detected. It is also shown that knowledge about the current scene context can be used to improve the lane detection performance.

Contents

Chapter 1

Introduction

1.1. Motivation

In the last decades, advanced driver assistance systems (ADAS) have become one of the most rapidly growing domain in automotive technology. While ABS (Anti-lock Braking System) and TCS (Traction Control System) have increased road safety for many years and already belong to the standard configuration of new vehicles, substantial advances in vehicle environment perception with several types of sensors allowed the market launch of modern ADAS, like *Adaptive Cruise Control*, *Automatic Parking*, and *Lane Keep Assist*. These systems are designed to increase the comfort of drivers and passengers, and can in particular cases even help to increase safety. By providing additional functionality to the vehicle, they get more and more important for buying decisions while it gets more difficult to convince with conventional topics like fuel consumption and emission only.

Although several passive and active safety systems (passive: e.g. airbag; active: e.g. ABS) have led to a decreasing number of injuries and fatalities, the statistics on accidents still show high potential to prevent accidents caused by human errors by making vehicles more intelligent. In 2014, road accidents in Germany led to 389,500 injuries and 3,377 fatalities. From the 302,400 accidents causing the mentioned physical injuries, 7.8% had general reasons related to road surface or sight conditions, or obstacles, and 0.9% can be assigned to technical issues of the vehicle. The remaining accidents were caused by human errors of the involved vehicle drivers, bicyclists or pedestrians, where 68.8% of these errors were committed by the vehicle drivers. These errors were mainly related to inadequate velocities and distances to other vehicles, right of way, and turning maneuvers[1].

[1] `https://www.destatis.de/DE/Publikationen/Thematisch/TransportVerkehr/Verkehrsunfaelle/VerkehrsunfaelleJ.html`

At this point, intelligent vehicles, equipped with advanced driver assistance systems or automatic driving functionality, can increase the traffic safety by preventing most of the errors of human drivers.

For a human, the eyes are by far the most important source of information for managing the complex task of driving. The technical equivalent, the camera, might be the sensor with the greatest potential for future vehicle environment perception systems. The capabilities of the human eye, with its wide angle of view and the ability to focus at a point of interest to gather information in near and far range, are not achieved by present camera systems. However, humans are able to perform the driving task just by watching camera images or simulated camera images in video games, displayed on common 2D screens. This shows that camera images provide enough information for driving, but the problem of extracting this information is still not solved. Powerful algorithms for the detection and classification of objects, or the segmentation of the image into regions already exist, but a complex task like driving a vehicle requires not only knowledge about locations of objects. This task requires an understanding of the current situation and the semantic meaning of the detected objects.

This work is a contribution to the way from image segmentation to scene understanding by classifying the current traffic scenario and analyzing the semantic meaning of different features in the vehicle environment in form of their spatial relations to the vehicle trajectory. This leads to a camera-based lane detection system relying on spatial relations of different types of features and contextual information about the scene.

1.2. Related Work

For the task of automatic driving or advanced driver assistance systems, knowledge about the location of the lane is required. There are two basic concepts for obtaining this knowledge. One possibility is to localize the vehicle inside a given map with defined lanes. The second way is to detect the lane by analyzing the sensor data, independently of any maps. However, especially in urban environments, precision and availability of GPS, as well as the correctness of the map, are not guaranteed. So, a sophisticated sensor data processing is crucial to reliable system performance. Huang et al. use the coarse location obtained by GPS as prior information for precise localization in a graph-based road map with laser and image processing (Huang et al., 2008). Kammel and Pitzer use the depth and intensity information from a LiDAR for curb detection (Kammel and Pitzer, 2008). With the result, the position inside a map obtained by GPS is refined. A coarse GPS-based localization can also be used to estimate the road geometry to support vision-based lane detection systems (Wang et al., 2004). For lane departure warning systems in highway scenarios, vision-based lane marking detection systems already show good performance.

DLD (dark-light-dark) transitions can be extracted from camera images to find the lane markings reliably (Felisa and Zani, 2010). Kluge determines the road curvature by detecting edges and calculating their orientation in the image (Kluge, 1994). The lane detection can be improved with the help of more complex features, e.g., vehicles (Lim et al., 2009).

Besides lane detection methods relying on the existence of lane markings, several approaches use texture analysis for image segmentation. For this purpose, a certain part of the image, directly in front of the vehicle, is assumed to belong to the road, and starting from the border of this area, deviations from the corresponding texture characteristics are detected. These methods usually classify the image pixels into road and non-road (Kuhnl et al., 2011; Gao et al., 2007). A texture-based image segmentation thus can be used also in unstructured environments without lane markings. However, texture analysis is often time consuming, and one disadvantage of these methods is that they can be affected by local structure deviations on the street, like dirt, holes, or surface changes. Furthermore, they usually do not provide a precise lane or lane border location in the image, and thus require further processing steps.
Important information about the course of the road in camera images from road scenes can be provided by the vanishing point. Kong et al. detect the vanishing point by calculating texture orientations with Gabor wavelets (Kong et al., 2010). Dominant lines through this point are then considered as the road borders. To apply complex texture analysis in real-time applications, specialized hardware for parallel processing can be used. Shang et al. extract edges pointing towards the vanishing point for lane detection using an FPGA (field programmable gate array).
The detected lane or lane border candidates, regardless of the detection method, are often compared to a previously defined model. These models allow the integration of prior knowledge about the appearance of the detected features. This can increase the performance and assures that a result follows certain constraints and thus avoids unrealistic solutions. While Alon et al. define the drivable area to be bordered by two parallel lines (Alon et al., 2006), Felisa and Zani model the lane border as a sequence of line segments (Felisa and Zani, 2010). More variation in the shape of the road is allowed by using more complex models. The road curvature in 2D can, e.g., be modeled with B-Snakes (Wang et al., 2004). Loose and Franke use B-Splines for modeling the road curvature even in 3D (Loose and Franke, 2010). The more variation a model permits, the more complex is in general the matching of the sensor data to this model, and the more the solution might be affected by disturbances in the data, like local texture changes and image noise.

To decrease the risk of sporadic wrong solutions, temporal consistence can be assured by smoothing the results with a Kalman filter (Paetzold and Franke, 1998). Also a particle filter can be applied to track the detected lanes over multiple frames (Linarth and Angelopoulou, 2011). A simpler approach to obtain more consistent results is to only allow solutions which do not differ too much from a previous result (Gao et al., 2007). Anyway, temporal tracking can increase the system performance and thus is very useful when an

algorithm is used in a real application, but it might also disguise the true performance of the underlying method.
Recently, Schomerus et al. presented a camera-based method for real-time lane border detection in arbitrarily structured environments (Schomerus et al., 2014). This approach is described in detail in section 6.6.
Besides conventional cameras, other sensors can be used to detect the road or its borders. A Velodyne LiDAR is used by Chen et al. for curb detection (Chen et al., 2015), and Peláez et al. detect the road with a thermographic camera (Pelaez et al., 2015).

1.3. Contribution

Existing approaches for lane detection use traditional segmentation methods for separating the road and non-road regions in the camera image, or detect a few typical lane boundary features like lane markings and curbstones. Most of them can be considered as local approaches, since they only analyze a certain area in the image or use prior knowledge about the free space location.

In this work, several local features in the whole camera image, typical of traffic scenes, are detected and classified (chapter 3). Furthermore, the context of the current scene is determined (chapter 5). Since information from the whole image is analyzed for feature detection and context classification, this method can be considered as a global approach. For the detected local features, the spatial relations to the vehicle trajectory are learned with a large training data set (chapter 2). And it is shown how different types and sources of information about the position of the lane can be combined (chapter 2).
The recorded vehicle trajectory serves as *virtual ground truth* for the course of the lane. The usage of this self-generated *virtual ground truth* makes it possible to apply powerful machine learning techniques which require a great amount of training data. With these learned spatial relations, a vehicle trajectory can be estimated which in turn can be used as an initial solution for a precise lane or lane border detection or trajectory generation (chapter 4).
The traffic scenes are divided into different context classes. By classifying the context of the current vehicle environment, the lane detection performance can be improved. For this purpose, different ways of taking advantage of the contextual knowledge are analyzed. Depending on the context, a certain lane matching method is chosen, spatial relations for each context class are learned, and a context-depending weighting of the feature classes is performed (chapter 6).

Concluding, the main contributions of this work are:

- Learning spatial relations between local features and vehicle trajectory
- Training with virtual ground truth
- Detection and classification of several traffic scene related features
- Trajectory estimation with learned spatial relations of local features
- Fusion of different sources of information about the position of the lane
- Context classification of traffic scenes with global features
- Improvement of lane detection methods by using context information
- Proposing the combination of global and local approaches for lane detection

Chapter 2

Virtual Ground Truth: Learning Spatial Relations

2.1. Virtual Ground Truth

Computer vision methods are often used to detect certain objects in camera images or to perform a segmentation of these images into regions with certain characteristics or semantic meanings. Which approaches are applied does not only depend on the problem, but also on the kind of information available in the software development phase. If the exact characteristics of an object are known and can be described, e.g., by a set of parameters, model-based matching approaches like RANSAC (Fischler and Bolles, 1981) or the Hough-Transform (Duda and Hart, 1972) are powerful tools for the detection of this object. If the characteristics are not known completely or show a high variety, machine learning approaches can be used to learn a corresponding model from a set of training samples. The segmentation of an image into regions of different textures without prior knowledge about those textures can already be considered as a simple machine learning task. The texture at one location of the image is analyzed and described by certain parameters, e.g., with Co-occurrence Matrices (Haralick et al., 1973) or Local Binary Patterns (Ojala et al., 1994), and for other image coordinates, it is decided whether those texture-describing parameters are similar or differ significantly.
If prior knowledge about the concerned object or texture is available, e.g., a set of training samples, it is possible to generate a model that describes the concerned entity, also containing the variety of characteristics (in the training data). If ground truth data are available for an offline learning phase, powerful mechanisms for this automated model generation can be applied. In this case, the performance of the detection or classification system does not depend on manually created model parameters, but on the amount and quality of ground truth training data. Unfortunately, in most tasks, this ground truth has

to be defined manually. This means that, in a set of training images, the entity of interest (object or image region) has to be labeled by a human.

Compared to many other vision-based object detection tasks, there is one essential difference in vehicle environment perception. A big effort is required to obtain training or test data, including the vehicle and camera setup, calibration, other sensors like IMU (inertial measurement unit), GPS (Global Positioning System), odometry, etc. But once the system is installed, a lot of data with a great variety can be obtained easily. Despite the great amount of data, these training images can be used to generate classifiers for certain objects or textures, if they are marked manually offline, or if a certain area in the image is generally defined as drivable area. So, e.g., the drivable area in the image, curbstones or pedestrians can be detected.
However, camera-based lane estimation is more than texture classification or object detection. The great variety in the appearance and semantic meaning of such objects in traffic scenes requires a sophisticated analysis of the relations between any entities (objects or other features) in the image and the path a vehicle should drive. Of course, the ground truth of the ego-lane can be labeled manually in a set of images, but creating a ground truth data set covering all relevant traffic scenarios will not be possible. At this point, the concept of virtual ground truth is introduced. Instead of manually labeling a great amount of data, the recorded vehicle trajectory in the training sequences can be used as a virtual ground truth. If the test vehicle has recorded odometry data, possibly supported by a GPS receiver or an inertial measurement unit (IMU), the trajectory for the following time steps can be projected into the current image, as depicted in figure 2.1. If now several features (objects, textures, etc.) are detected, the spatial relations between these local features (with a certain location in the image) and the vehicle trajectory can be learned. This way, the problem of missing ground truth is overcome by an automatic generation of virtual ground truth.
The next section will describe how these spatial relations are modeled and how they can be used for lane estimation.

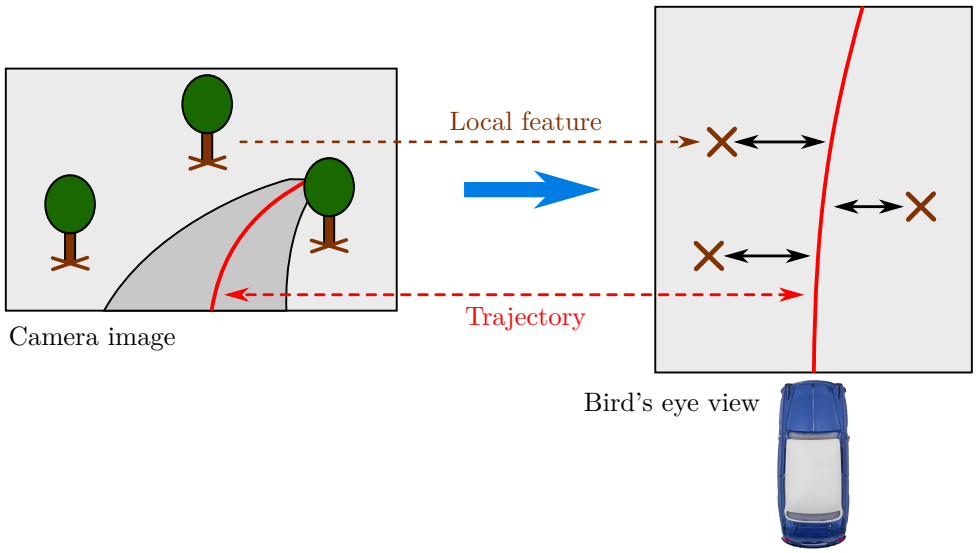


Figure 2.1.: Recorded trajectory in camera image and bird's eye view, with local features and the lateral offsets to the trajectory

2.2. Modeling Spatial Relations

Since the modeling of spatial relations requires information about the location of objects (or other entities) with respect to the vehicle trajectory, features with certain pixel coordinates in the image are considered. In chapter 3, these features are described in detail. In this section, it is only important that these local features have a certain position. In the perspective view of the camera images, the spatial relations between objects depend on the position within the image or rather on the distance to the camera. Since metrical positions are required to model the spatial relations within the scene, it is reasonable to work in the bird's eye view, which is a projection of the camera image onto the ground plane. In this projected image, a certain number of pixels corresponds to a certain distance in meters, so metrical information can be used, at least for objects or features lying on the ground plane. To create this projected image, the relation between camera and ground plane has to be known. This section describes the required coordinate systems of the vehicle, the creation of the bird's eye view, and the representation of spatial relations.

2.2.1. Coordinate Systems of the Vehicle

To calculate spatial relations between the vehicle trajectory and features, some coordinate systems have to be defined. Figure 2.2 shows the coordinate systems V for the vehicle base, G for the ground plane, C for the camera, and P for the position capture unit.

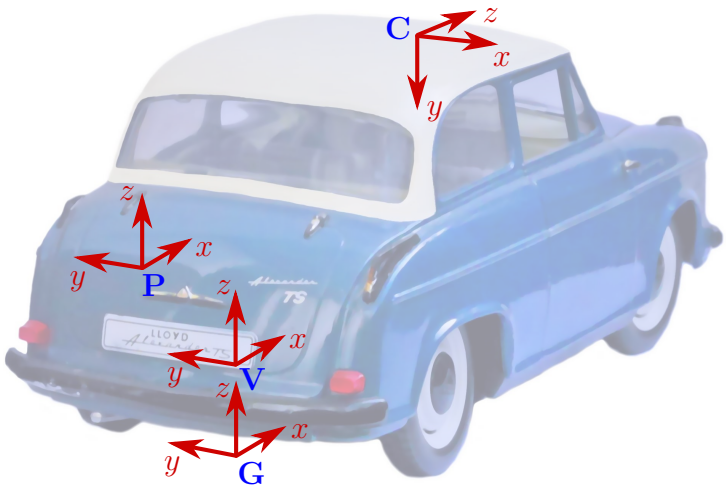


Figure 2.2.: Vehicle coordinate systems

These coordinate systems are listed in table 2.1.

Table 2.1.: Coordinate systems of the vehicle

V	Vehicle base coordinate system
G	Ground coordinate system
C	Camera coordinate system
P	Coordinate system of the position capture unit

The vehicle base coordinate system V is typically located in the center of the rear axis, with x pointing in driving direction, y to the left and z pointing upwards. The ground coordinate system G is lying on the ground plane in negative z direction of V, with x pointing in driving direction, y to the left and z pointing upwards, perpendicular to the ground. The camera coordinate system C has its origin in the optical center (focal point) of the camera, with z as optical axis (in viewing direction), x pointing to the right and y downwards. To capture the trajectory of the vehicle, different techniques can be used and combined to increase the accuracy. The odometry of the vehicle can be used and enhanced, e.g., with an inertial measurement unit (IMU) or even with GPS information. Regardless of which technique is used to capture the position or motion of the vehicle, the device will be referred to as the position capture unit here and is assigned to the coordinate system P.

To describe the transformations between these coordinate systems, homogeneous transformation matrices are used. These (4×4) matrices have the advantage that the translational

part of the transformation can be directly read from the fourth column and that multiple transformations can be concatenated by simply multiplying the matrices. Furthermore, the inverse transformation can be calculated easily. The transformation

$$ {}^{A}\mathbf{T}_{B} = \begin{pmatrix} \mathbf{R} & t \\ 0^{\top} & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.1} $$

describes the pose (position and orientation) of coordinate system B with respect to coordinate system A. $\mathbf{R}$ is an orthogonal (3×3) rotation matrix which describes the orientation, t is a 3D translation vector and $0^{\top}$ is a horizontal (transposed) 3D zero vector.

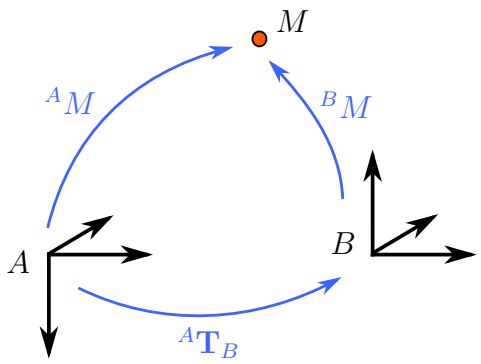


Figure 2.3.: Transformations between coordinate systems

The position of a 3D point $M = (x, y, z)^{\top}$ can now be specified with respect to any coordinate system. ${}^{A}M$ describes the point in the coordinate system A and ${}^{B}M$ is the same point in the coordinate system B, which is illustrated in figure 2.3. With $\tilde{M} = (\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w})^{\top} = (x, y, z, 1)^{\top}$ being the same point M in homogeneous coordinates, the coordinates of M with respect to B can be translated into the coordinate system A and vice versa by

$$ {}^{A}\tilde{M} = {}^{A}\mathbf{T}_{B} \cdot {}^{B}\tilde{M} \quad \text{and} \quad {}^{B}\tilde{M} = {}^{B}\mathbf{T}_{A} \cdot {}^{A}\tilde{M} = ({}^{A}\mathbf{T}_{B})^{-1} \cdot {}^{A}\tilde{M} \tag{2.2} $$

and the inverse of ${}^{A}\mathbf{T}_{B}$ is

$$ {}^{B}\mathbf{T}_{A} = ({}^{A}\mathbf{T}_{B})^{-1} = \begin{pmatrix} \mathbf{R}^{\top} & -\mathbf{R}^{\top} t \\ 0^{\top} & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & -t \cdot r_1 \\ r_{21} & r_{22} & r_{23} & -t \cdot r_2 \\ r_{31} & r_{32} & r_{33} & -t \cdot r_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3} $$

where r_i is the i-th column vector of $\mathbf{R}$. The coordinates of M can be retrieved from the homogeneous point $\tilde{M}$ by $M = (\tilde{x}/\tilde{w}, \tilde{y}/\tilde{w}, \tilde{z}/\tilde{w})^{\top}$.

Note that beside this form of describing transformations between coordinate systems, another form can be found in the literature. If the transformation of the coordinates of a point M from A to B is specified without homogeneous transformation matrices by ${}^{B}M = \mathbf{R}' \cdot {}^{A}M + t'$, then $\mathbf{R}'$ and t' are the elements of the inverse transformation matrix ${}^{B}\mathbf{T}_{A}$ and correspond to

$$\mathbf{R}' = \mathbf{R}^{\top} \quad , \quad t' = -\mathbf{R}^{\top}\, t \quad \text{and} \quad t = -\mathbf{R}'\, t' \,. \tag{2.4}$$

Although this relation is a simple consequence of the matrix multiplication, the different ways of describing transformations in the literature should be kept in mind especially when different (public) datasets are used.

To create the bird's eye view of the camera image, and to analyze the spatial relations between features detected by the camera and the vehicle trajectory recorded by the position capture unit, the transformations between the coordinate systems V, G, C and P have to be measured.

2.2.2. Bird's Eye View

The expression *bird's eye view* (BEV), or *top view*, describes the result of mapping the camera image to the ground plane, which gives the impression of a view on the scene from above. This process is often referred to as an inverse perspective mapping (Mallot et al., 1991). While the perspective mapping is the projection of a 3D point $M = (x, y, z)^{\top}$ to the image plane, resulting in a 2D point $m = (x, y)^{\top}$, the inverse perspective mapping describes the projection of an image point m back into the 3D space. In order to calculate these projections, it is necessary to know the camera characteristics. These characteristics are described by the extrinsic and intrinsic camera parameters, where the extrinsic parameters define the pose of the camera projection center with respect to, e.g., a world coordinate system. The intrinsic parameters describe the physical composition of the camera which defines the projection characteristics. These parameters depend on the model which is used to describe the camera. They specify, e.g., the focal length, the principal point in the image (intersection of the optical axis with the image plane) and the lens distortion. The determination of the extrinsic and intrinsic parameters of a camera is the camera calibration. Common camera models and corresponding calibration methods are, e.g., the Tsai Camera Model (Tsai, 1986) and the Zhang Camera Model (Zhang, 1999). These models are based on pinhole cameras with a central projection but also model the lens distortion. With a valid calibration of the camera, the image distortion by the lens can be corrected, so the projection can be described by a projection matrix

$$\mathbf{P} = \mathbf{A}\,[\mathbf{R}' \,|\, t'] = \mathbf{A}\,[\mathbf{R}^{\top} \,|\, -\mathbf{R}^{\top} t] \tag{2.5}$$

with the calibration matrix

$$\mathbf{A} = \begin{pmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.6}$$

with the principal point $(p_x, p_y)^\top$ and the focal length f in pixels. For the case of non-quadratic pixels on the sensor, f is specified in horizontal and vertical pixel dimensions, so $f_x = f_{mm} \cdot P_{x/mm}$ with f_{mm} = focal length in mm and $P_{x/mm}$ = pixels per mm in x-direction, and f_y for pixels in y-direction, respectively. The origin of the image coordinate system, the image point $(0, 0)^\top$, is located in the upper left corner of the image. In this model, a right angle between the x- and y-axis on the sensor is presumed.

Note that, if the coordinate system C of the camera is specified with a homogeneous transformation matrix $\mathbf{C}$, $\mathbf{R}'$ and t' correspond to the rotation matrix and the translation vector of the inverse matrix $\mathbf{C}^{-1}$, as described in equation 2.4 in section 2.2.1.

The projection of a 3D point M in homogeneous coordinates $\tilde{M} = (\tilde{M}_x, \tilde{M}_y, \tilde{M}_z, \tilde{M}_w)^\top$ to the 2D point $\tilde{m} = (\tilde{m}_x, \tilde{m}_y, \tilde{m}_w)^\top$ is then defined by

$$\tilde{m} = \mathbf{P}\, \tilde{M} \tag{2.7}$$

and the (non-homogeneous) image coordinates of the projected point are retrieved by $m = (\tilde{m}_x/\tilde{m}_w, \tilde{m}_y/\tilde{m}_w)^\top$. A point ${}^C M$, already given in the coordinate system of the camera, is projected by a multiplication with just the calibration matrix extended by a fourth column filled with zeros:

$$\tilde{m} = [\mathbf{A} \mid (0, 0, 0)^\top] \cdot {}^C\tilde{M}\,. \tag{2.8}$$

This corresponds to equation 2.7 with no rotation ($\mathbf{R} = \mathbf{R}'$ = identity matrix) and translation ($t = t' = (0, 0, 0)^\top$). Figure 2.4 illustrates this projection.
Since one dimension is lost during the mapping of 3D points to the image plane, for the back projection to the scene, further knowledge is necessary. If the transformation ${}^C\mathbf{T}_G$ between the camera coordinate systems C and the ground G (see section 2.2.1) is known, which defines the ground plane in camera coordinates, the camera rays (lines from the origin of C through the centers of the pixels on the sensor) can be intersected with this plane.

Figure 2.5b shows the result of this inverse perspective mapping of the camera image in figure 2.5a. This procedure is useful when the position of certain pixels, e.g., features detected in the camera image, are needed in vehicle coordinates. For the creation of a full bird's eye view image with a given resolution, it is more reasonable to use the other projection direction, described in equation 2.8, because the pixel density (pixels in camera image per m^2 in the BEV image) decreases rapidly with growing distance from the camera. Then, the pixel positions of the BEV image can be expressed as 3D points

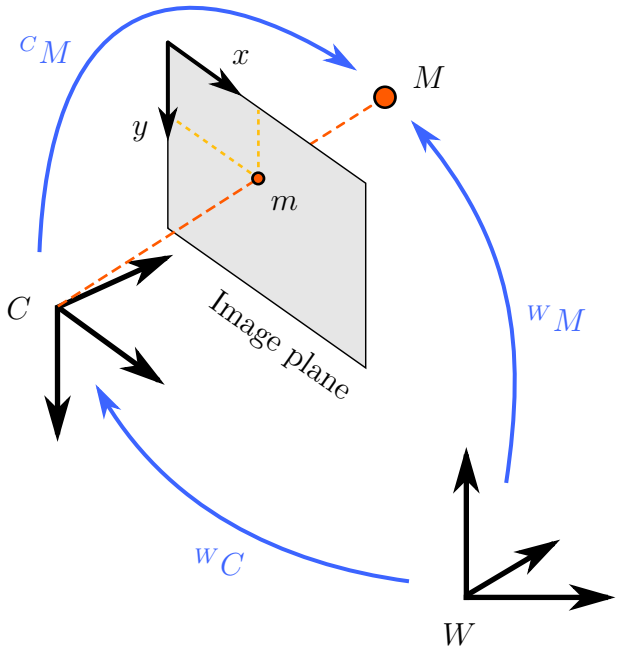
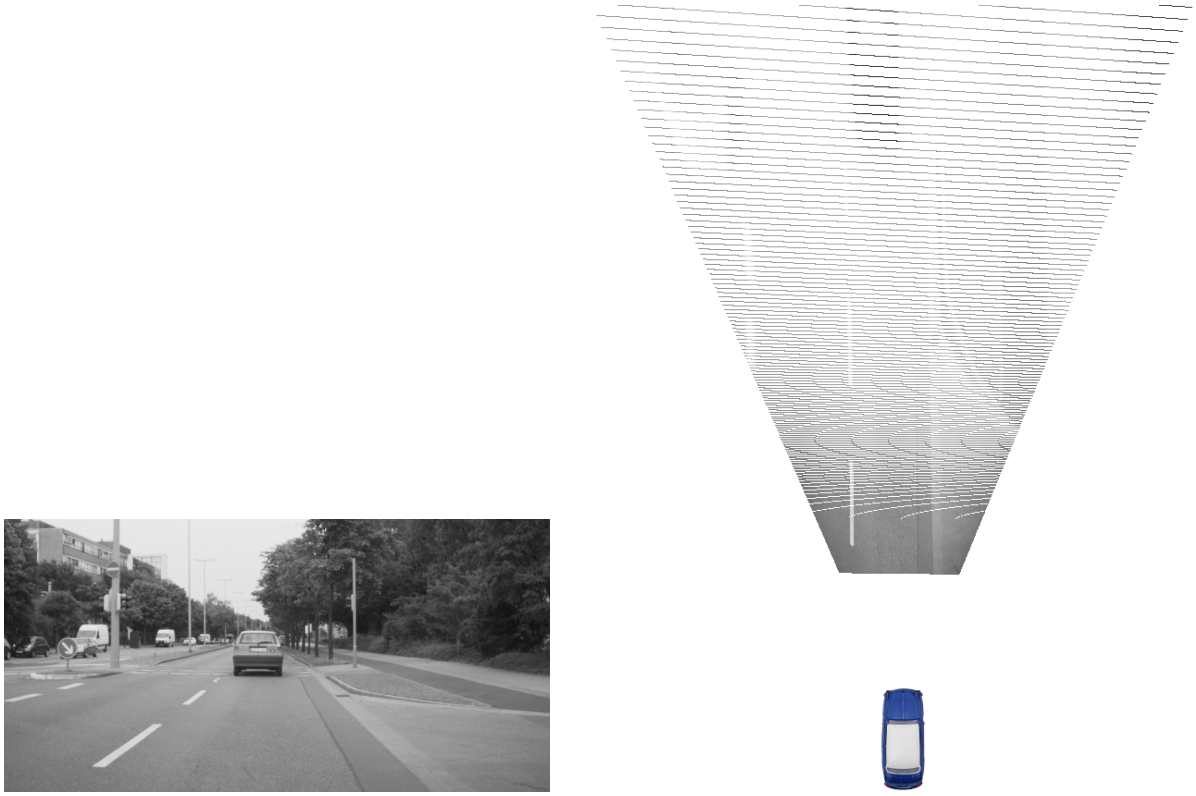


Figure 2.4.: Projection of a point M to the image plane of the camera C

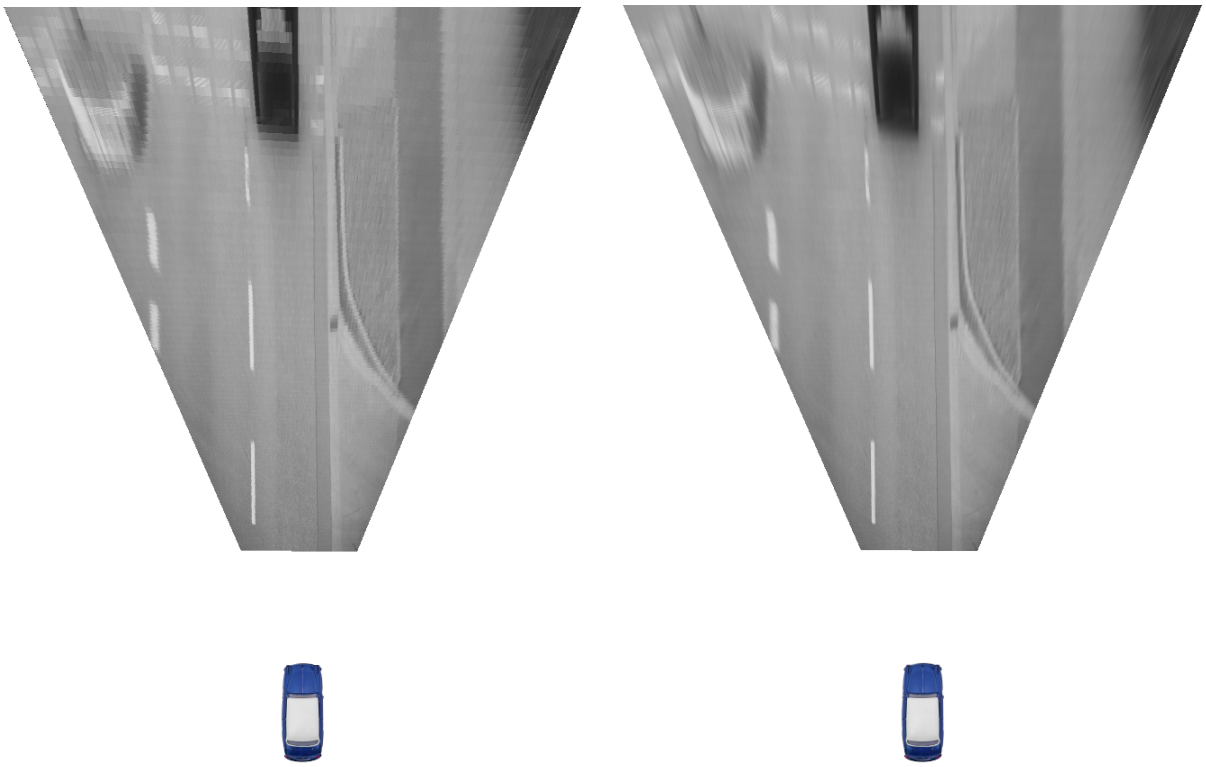
with respect to the ground coordinate system, transformed into the camera coordinate system by $^{C}\tilde{M} = {}^{V}\mathbf{T}_{C}^{-1} \cdot {}^{V}\mathbf{T}_{G}$, and projected to the image plane according to equation 2.8. This way, the gaps with empty pixels which are not intersected by the camera rays can be avoided, because for each pixel of the BEV image a corresponding pixel in the camera image exists.

The artifacts in the upper region of figure 2.6a arise because several neighboring pixels in the BEV image are projected to the same single pixel of the camera image. If the BEV image is used for feature or object detection, tracking or classification, these artifacts will disturb the procedure, because they make the appearance of an object in the far range differ essentially from the same object close to the camera. To obtain a BEV image with higher quality also in the far range, the BEV pixels can be projected to the camera image with subpixel precision. With this subpixel location in the camera image, the best pixel value can be calculated by interpolating between the neighboring pixels. Figure 2.6b shows the result of the mapping with bilinear interpolation (linear interpolation in both x- and y-direction of the image). In such a bird's eye view image, the features can be detected and classified accurately, almost independently of the distance to the camera, as will be described in chapter 3. Anyway, a maximum distance from the camera (and so the size of the BEV image) has to be chosen to get a BEV image of adequate quality.



(a) Camera Image **(b)** Inverse perspective mapping

Figure 2.5.: Camera image and projection with inverse perspective mapping



(a) Choosing nearest camera pixel **(b)** Bilinear Interpolation

Figure 2.6.: Projection of the camera image to the ground plane. The artifacts are reduced by bilinear interpolation.

2.2.3. Representation of the Spatial Relations

The previous section described the transformation between image and vehicle coordinates. With these transformations, the features, which are detected at certain locations of the camera image or in the bird's eye view image, obtain a corresponding position with respect to the vehicle. And the location of such a feature is described as a 2D point on the ground plane. The vehicle trajectory, on the other hand, can be an arbitrarily parameterized curve. And for every frame of the learning data, the considered curve, which represents the future vehicle positions, can be different. Of course, it would be possible to model the spatial relations between feature points and arbitrary curves. But regarding the expected feature types in the vehicle environment (lane markings, curbstones, trees, reflector posts, etc.), advantage can be taken of the fact that many appearing structures follow the course of the lane. So, the relation between a feature and the trajectory does very often not depend on the curvature of the lane. A curbstone (or a pixel classified as curbstone), e.g., only has a certain distance to the trajectory which is the same for straight and curved roads. Therefore, it is sufficient to model the spatial relation as the lateral offset between these features and the vehicle trajectory. So, the spatial relations between any interesting structure in the image and the vehicle trajectory can be simplified to a scalar representing the lateral offset. As illustrated in figure 2.7, for a precise calculation of this offset, the curvature of the trajectory would need to be considered.

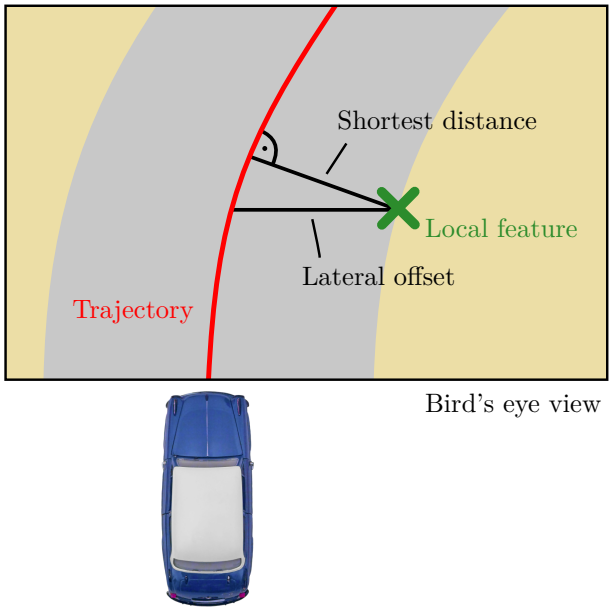


Figure 2.7.: Relation between the lateral offset and the shortest distance

In the training phase, the shortest distance between a feature and the trajectory can be calculated easily. But in the testing phase, when the unknown lane has to be estimated, the

location of the lane with respect to the feature turns from a lateral offset to a circular path around the feature with the radius according to the learned shortest distance. However, the difference between the lateral offset and the shortest distance is relatively small except for very narrow curves. This error or uncertainty can also be considered by a training phase with a high number of features and trajectories with different curvatures. So the curvature can be disregarded which leads again to just scalar lateral offsets.

When the lateral offset is learned with the virtual ground truth trajectory with a representative dataset, a high number of values is calculated. Since a feature (e.g., lane marking) does not have one possible offset, but several positions relative to the trajectory (e.g., on the left and on the right side of the lane), one value is not enough to represent the spatial relation. E.g., the average of the calculated offsets would not contain enough information. A representation of the spatial relation considering different values is necessary. Although the values are real valued, they can be saved in form of a histogram if a resolution has been defined previously. If, e.g., a precision up to 10 centimeters is considered to be enough, every bin of the histogram represents a region of 10 centimeters. An example of a lateral offset histogram is shown in figure 2.8. This histogram has 400 bins and a resolution of 20 bins per meter. The additional (orange) horizontal axis shows the corresponding offsets in vehicle coordinates in meters (the y-axis of the vehicle coordinate system points to the left). In this case, 2 meters to the right from a detected feature appears to be the highest probability for the vehicle trajectory. The histogram is created by collecting the offsets of all features in all frames available in the training dataset. Since this histogram contains the distribution of lateral offsets, it is also referred to as the *lateral offset histogram*.

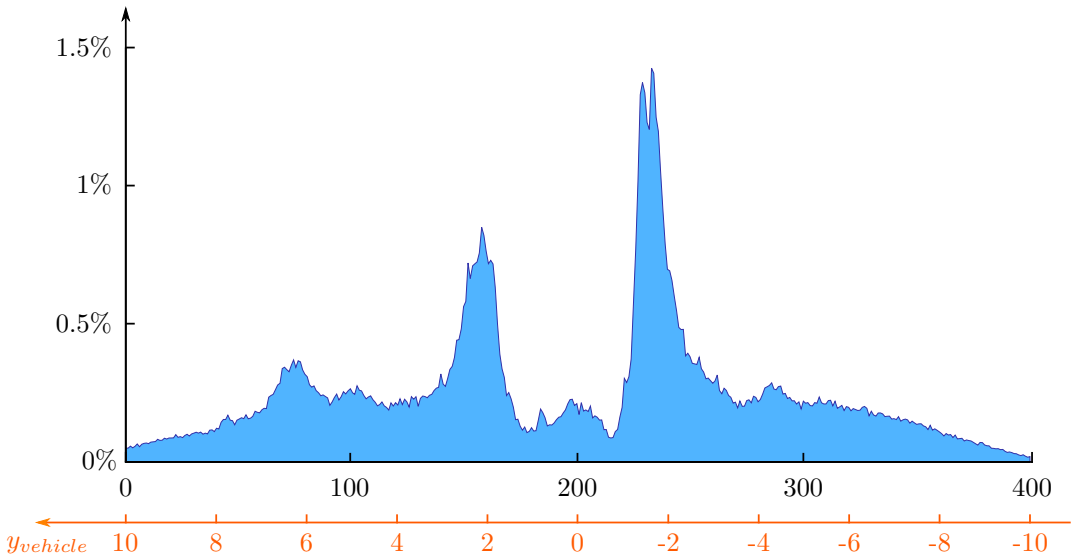


Figure 2.8.: The lateral offset histogram with 400 bins. The second (orange) horizontal axis shows the corresponding offsets in vehicle coordinates in meters (the y-axis of the vehicle coordinate system points to the left).

The offsets are defined in the vehicle coordinate system. The center bin of the histogram corresponds to an offset of zero on the y-axis. The first bin represents the greatest positive offset (trajectory on the left side of the feature). The advantage of this sample-based representation with a histogram is that it allows a fast training, since for each calculated offset only the corresponding cell in the histogram has to be incremented. The need of memory is small compared to the amount of analyzed data, and furthermore, it is predictable. The time needed for a normalization of the histogram at the end of the learning phase does not depend on the number of features, but only on the predefined resolution of the histogram. This predefined resolution is also a disadvantage of this approach. Because, if a higher resolution is needed, the whole learning phase has to be repeated or the missing information has to be interpolated. Also, a maximum offset between feature and trajectory has to be specified to define the size of the histogram. Anyway, this offset is limited by the width of the BEV image. Another possibility is to save all offsets from the training phase and use them later to create a histogram of the desired resolution. If a continuous representation of the spatial relations is desired, the saved values can be used to create a mixture model, which is a combination of different (e.g., Gaussian) distributions. Such a mixture model can be created by *Expectation-Maximization* (Dempster et al., 1977) or *Kernel Density Estimation* (Parzen, 1962). In the case of Expectation-Maximization, several Gaussian distributions with different parameters (mean μ, variance σ^2 and weight w) are combined to a probability density function

$$f(x) = \sum_{i=0}^{n} w_i \, g(x; \mu_i, \sigma_i^2) \tag{2.9}$$

with $\sum w_i = 1$ and the kernel (here a univariate Gaussian distribution)

$$g(x, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \, \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}} \, . \tag{2.10}$$

The parameters are refined iteratively to maximize the log-likelihood. The disadvantage of this method is that the number n of combined distributions has to be chosen and that the iterative refining of the parameters is prone to find local maxima. The Kernel Density Estimation, on the other hand, generates a (e.g., Gaussian) kernel function for every sample value. All these kernels are also combined as in equation 2.9, but with fixed weights $w_i = w$ and $\mu_i = x_i$. The remaining parameter $\sigma_i = \sigma$ (in Kernel Density Estimation referred to as the *bandwidth* of the kernel) is chosen depending on the underlying data and has a strong influence on the quality of the result (a higher value of σ leads to a smoother curve). However, there are approaches for tuning this parameter automatically (Sheather and Jones, 1991). Since a kernel is created for every sample of the data, a subsequent manipulation (sampling, evaluation, etc.) becomes time-consuming if a high number of values has been used for training. For this reason, a continuous representation should only be created if necessary, e.g., if a further calculation step requires real (continuous) probability densities. Here, it is more feasible to use a sample-based distribution in form of a histogram, which does not only allow a fast training, but also a fast sampling of the underlying distribution, as shown in the next section.

2.3. The Distribution Map

The previous section described the result of the learning of spatial relations. The lateral offsets between features and vehicle trajectories are saved in form of a histogram of predefined resolution. Such a histogram represents the distribution of trajectory locations relative to the feature. In the field of lane detection, these learned spatial relations can be used to estimate the most probable vehicle trajectory. The estimated trajectory can then be considered as the center of the lane and can be used as a starting location for searching the lane borders. For this reason the estimated trajectory also represents the lane.

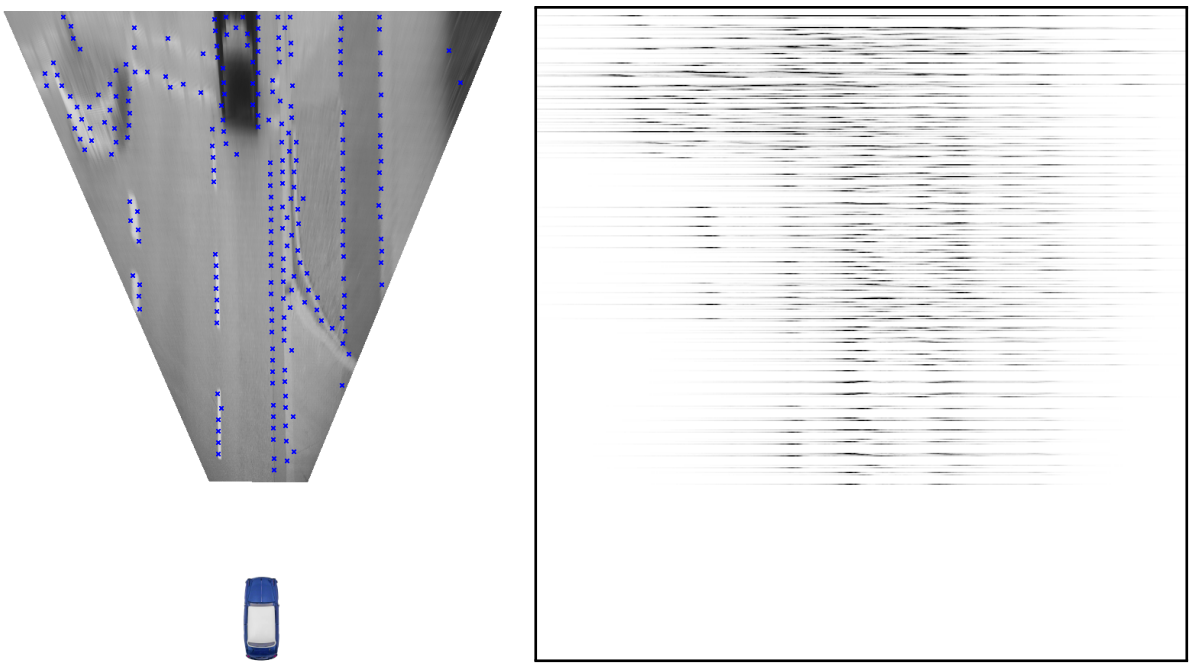
2.3.1. Registration of Spatial Relations

To estimate the best matching vehicle trajectory, a map of the vehicle environment is created, approximately according to the analyzed image area (bird's eye view). For every feature detected in the camera or BEV image, the offset distribution, represented by the histogram, is plotted into this map at the location of the feature. So, the distribution map is generated by registering the previously learned spatial relations of all detected features. The pixels of this map correspond to the pixels of the BEV image and thus to a certain position with respect to the vehicle. If the learned histograms of lateral offsets are considered as probability density functions, then every bin of the histogram contains a value which is closely related to its possibility to belong to the trajectory. If the resolution of the histogram is chosen equally to the resolution of the distribution map, one pixel of the map and one bin of the histogram correspond to the same distance in meters, and the update of the distribution map $\Psi(x, y)$ is performed according to the following code:

```
foreach feature f at coordinates (f_x, f_y)
    foreach bin b in histogram h of f
        Ψ(f_x − h_halfWidth + b, f_y) = Ψ(f_x − h_halfWidth + b, f_y) + h_b
```

Note that the center bin of the histogram, $h_{halfWidth}$, corresponds to an offset of zero and to the possibility of a trajectory at the location of the feature. The advantage of the sample-based representation of the spatial relations with a histogram is that, instead of sampling the underlying distribution, the distribution map is updated by just plotting the histograms into the map. By inserting the histograms into the map, the 1D probability densities are extended by a second dimension according to the height of a row, which is the height of a pixel in the map. So, the distributions, which can be seen as mixtures of Gaussians, are combined to a 2D mixture of 1D mixtures of Gaussians. This is why the insertion of the histograms is not proceeded by multiplication of probabilities, as for a joint distribution of independent random variables. The resulting map can be considered

as a potential field, where the best trajectory will be fitted into. Figure 2.9a shows some exemplarily detected features (the feature detection will be described in chapter 3), and figure 2.9b shows the corresponding distribution map, where for each feature the lateral offset histogram is registered.



(a) Detected features **(b)** Registered spatial relations

Figure 2.9.: The distribution map: values range from 0 (white) to 1 (black)

Of course, different types of features have different spatial relations to the trajectory. While in this chapter the spatial relations between the vehicle trajectory and any abstract features are described, chapter 3 will discuss the different feature classes. It is important that the histograms are normalized, not only because they are considered as probability densities, but also because their influence on the distribution map should not depend on the number of training samples, especially if different types of features are used or other information sources are available which might improve the lane detection performance. How such additional information can be used will be described in the next section.

2.3.2. Integration of Additional Information: The Prior

The distribution map is not only suitable for collecting the lateral offset histogram entries of the detected features. It is also a powerful tool for combining various sources of information. In addition to the features and their learned spatial relations to the vehicle

trajectory, other information might be available, e.g., locations of obstacles from a radar or laser scanner, or the expected approximate driving direction from a road map. Besides, in the absence of additional sensors, one might want to prioritize, e.g., a forward direction. For this purpose, another distribution map can be created with higher values at the pixels in driving direction, as seen in figure 2.10a. Such a distribution map containing additional information to the spatial relations is referred to as a *prior*, since it is related to a prior probability distribution. To combine multiple distribution maps and control the influence of the individual maps on the result, these maps should be normalized, i.e.,

$$\iint \Psi(x, y)\, \mathrm{d}x\mathrm{d}y = 1\,, \tag{2.11}$$

so the resulting distribution map Ψ_{res} is defined by

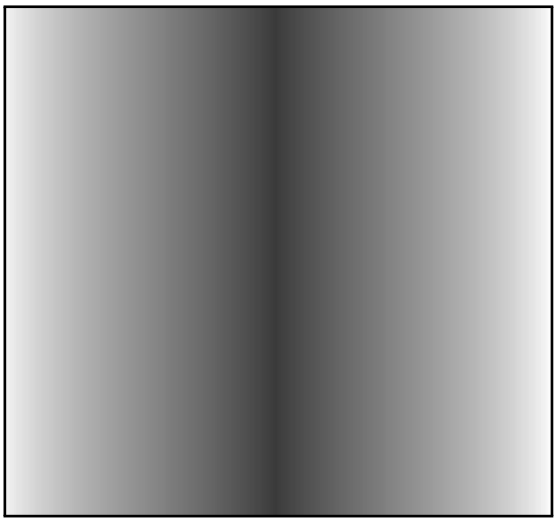
$$\Psi_{res} = \sum_i w_i\, \Psi_i\ , \tag{2.12}$$

where Ψ_i are the individual maps and w_i the weights defining their influence on the result with $\sum_i w_i = 1$. Combining, e.g., the distribution map of the features Ψ_{feat} and their spatial relations with a direction prior (a map prioritizing a forward driving direction) Ψ_{dir} and an obstacle prior (a map considering obstacles) Ψ_{obs}, leads to

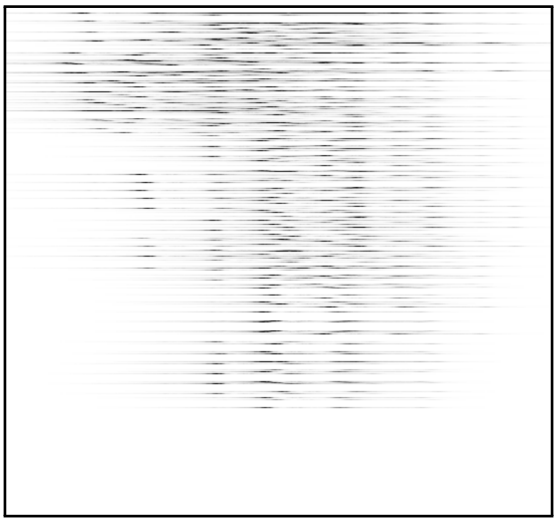
$$\Psi_{res} = w_{feat}\, \Psi_{feat} + w_{dir}\, \Psi_{dir} + w_{obs}\, \Psi_{obs} \tag{2.13}$$

with $w_{feat} + w_{dir} + w_{obs} = 1$. Figure 2.10a shows a possible distribution map which gives a higher priority to a forward driving direction. Static obstacles might be detected by other sensor systems, e.g., laser scanner or radar, which make it also possible to track dynamic obstacles over time, so a temporary stopping of moving vehicles, e.g., at traffic lights, does not disturb the prior map for static obstacles. Dynamic obstacles, e.g., other vehicles, could also be integrated, but since a moving vehicle might be located on the own lane, only static obstacles should be used to exclude the corresponding regions, i.e., to decrease the values in the distribution map. The resulting distribution map, which is a combination of the feature distribution map and the priors (here only the forward direction prior), is shown in figure 2.10c.

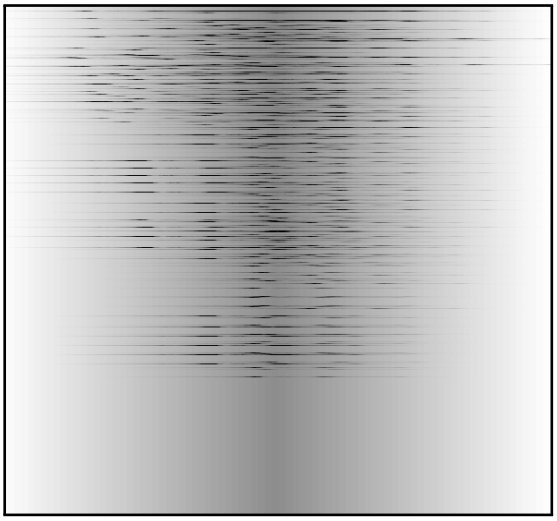
In this chapter, the generation of the distribution map with learned spatial relations and detected features was explained. Before chapter 4 describes how the distribution map is used to find the vehicle trajectory, the next chapter will show how the required features are detected and how different feature classes are distinguished to achieve feature class specific spatial relations and thus increase the quality of the distribution map.



(a) Prior for forward driving direction



(b) Original distribution map



(c) Combined distribution map

Figure 2.10.: Combination of prior and learned spatial relations

Chapter 3

Local Features: What Tells us Where to Drive the Car?

Camera images contain a great amount of information about the observed scene. However, it is hard to determine which part of the information is really necessary for finding the lane. Only considering the ideal case of an empty highway with dark asphalt and white lane markings in perfect road, weather and lighting conditions is not enough in practice. In fact, the appearance of road, lane and their borders differs extremely. However, there are several objects or patterns in the image, which provide some information about the possible location of the lane. While a detected tree attests the absence of drivable area at its location, a lane marking increases the probability of a lane, at least close to it. A human knows where to drive a car, because his experience already contains the semantic meaning of the objects, the correlation between the locations of certain objects and the lane. A machine has to learn these relations, too. And this is the learning of the spatial relations between features and the vehicle trajectory, as described in the previous chapter. As the example of the tree and the lane marking shows, these spatial relations are different for the different types of features. Thus, for every type of object or pattern, these spatial relations have to be learned separately. While the tree in the example above is already a quite complex feature, the lane marking, on the other hand, is in general easier to detect automatically. In this chapter, the different types of features helping to estimate the lane are presented. It also describes how these features are detected and classified. All these features are referred to as local features, since they possess a certain location in the image and in the vehicle environment.

3.1. Local Features

Local features are points of interest in the image, which promise help for solving the given task, e.g., finding the lane. Feature types can be divided into two categories: concrete features and abstract features. The category of concrete features, or semantic features, contains all patterns and objects with a known semantic meaning to the human, e.g., lane markings, curbstones, reflector posts, traffic signs, etc. Abstract features, or geometric features, are any patterns in the image without a prior meaning, e.g., edges, lines, corners, color patterns, etc. They are only defined by geometrical characteristics. Of course, every pixel of the image could be interpreted as an abstract feature, but usually geometrically significant patterns, like edges or corners, are searched for to restrict the number of features. These significant structures are expected to provide more information than regions with equal color or gray values.
This section describes the detection and the different types of these points of interest, the local features.

3.1.1. Feature Detection

Corners and edges are significant points in the image and many approaches exist to detect them efficiently. Some feature related tasks, e.g., feature tracking for *structure from motion*, require exact feature positions. So, for these tasks, often corner detectors like the FAST Corner Detector (Rosten and Drummond, 2005) are used. However, in the field of lane detection, many important features are only represented by an edge, e.g., lane markings, curb stones, etc. And as already described in section 2.2.3, more important than exact 2D coordinates of a feature is the lateral offset between feature and vehicle trajectory. While many significant points in the image would not be found by a corner detector, an edge detector would provide much more features which are important for the lane detection task.

A reliable method to detect edges in a gray value image is the Canny Edge Detection (Canny, 1986). This edge detector was designed to meet the following requirements:

- Good detection performance: detect as many edges as possible by minimizing the number of false positives
- Good localization: the location of the detected edge corresponds to the center of the edge
- One edge leads to one detector response (no additional responses close to the edge)

The Canny Algorithm, which is configured by 3 parameters, consists of the following steps:

1. Noise reduction
2. Gradient calculation (strength and direction)
3. Non-maximum suppression
4. Contour tracking

To reduce the noise, the image is convolved with a filter kernel which approximates a Gaussian function. Depending on the expected signal-to-noise ratio, the dimension n of the filter kernel can be chosen appropriately. This kernel size is the first parameter of the Canny Edge Detection, presuming the use of quadratic kernels. For n=3, the smoothing kernel

$$\mathbf{K}_n = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.1}$$

can be used. For the gradient calculation, the partial derivatives are determined. The partial derivative of an image $I(x, y)$ in x direction,

$$G_x(x, y) = \frac{\delta I(x, y)}{\delta x} = \frac{I(x + 1, y) - I(x - 1, y)}{2}\,, \tag{3.2}$$

corresponds to the strength of an edge in y direction (vertical edge) at the pixel (x, y), and the partial derivative in y direction,

$$G_y(x, y) = \frac{\delta I(x, y)}{\delta y} = \frac{I(x, y + 1) - I(x, y - 1)}{2}\,, \tag{3.3}$$

corresponds to the strength of an edge in x direction (horizontal edge). The gradient strength at location (x, y) is defined by

$$G(x, y) = \sqrt{G_x^2 + G_y^2}\,, \tag{3.4}$$

and the gradient direction is determined by

$$\Theta_G(x, y) = \text{atan2}(G_y, G_x)\,, \tag{3.5}$$

where atan2(a, b) is the two-argument variant of the arctangent function.

In some implementations of the Canny Edge Detector, the first two steps (noise reduction and calculation of the partial derivatives) are performed by applying the Sobel Operator, since the Sobel filter kernels consist of a differentiation and a smoothing part perpendicular to the derivative direction, e.g., for G_x:

$$\mathbf{S}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} . \tag{3.6}$$

The third step is the non-maximum suppression. To assure that one edge in the image leads to only one detector response, at every pixel location the gradients of the neighboring pixels in positive and negative gradient direction are analyzed. If one of the neighboring pixels in gradient direction (perpendicular to the edge) has a higher gradient strength, then the gradient value of the analyzed pixel is set to 0. By this procedure, only the strongest gradient of a gray value transition is preserved and surrounding non-maxima are suppressed.

In the last step of the algorithm, connected edge pixels are privileged. For this purpose, two thresholds t_{low} and t_{high} with $t_{low} < t_{high}$, which are the remaining parameters of the algorithm, are used to label the pixels as *weak* ($t_{low} < G(x, y) < t_{high}$) or *strong* ($G(x, y) > t_{high}$) edge candidates. The idea is to delete single spurious weak gradients which might have been arisen from image noise. So, for every pixel labeled as *weak*, the 8-neighborhood is analyzed. If it is connected to a *strong* edge pixel, it is preserved as an edge pixel, otherwise, it will be suppressed.

Because of its good performance regarding the requirements given above, the Canny Edge Detector has become a very popular algorithm for edge detection. Figure 3.1 shows the result of the Canny Edge Detection algorithm on the bird's eye view image of a typical urban traffic scene.

Regarding the feature characteristics in the task of lane detection, as discussed in the beginning of this section, the feature selection can be improved by preferring edges in driving direction. These features do not only have a higher probability to define the border of the lane or the road, but are also less affected by image blur even at high velocities, which simplifies the detection. If these edges are preferred in the detection phase, the result would be less dependent on the vehicle velocity. So, inside the Canny Edge image, line segments in (approximate) driving direction are searched for. Since the image is projected onto the ground plane, the corresponding line segments are vertical lines in the image. Such a line segment is chosen as a feature, if it has a certain length. To consider also interrupted lines and different gradient values, three parameters have been defined: These are a minimum line length, a minimum edge value, and a minimum line portion. The minimum line portion specifies which portion of the line segment (of the minimum line length) needs to have at least the given minimum edge value. If a line

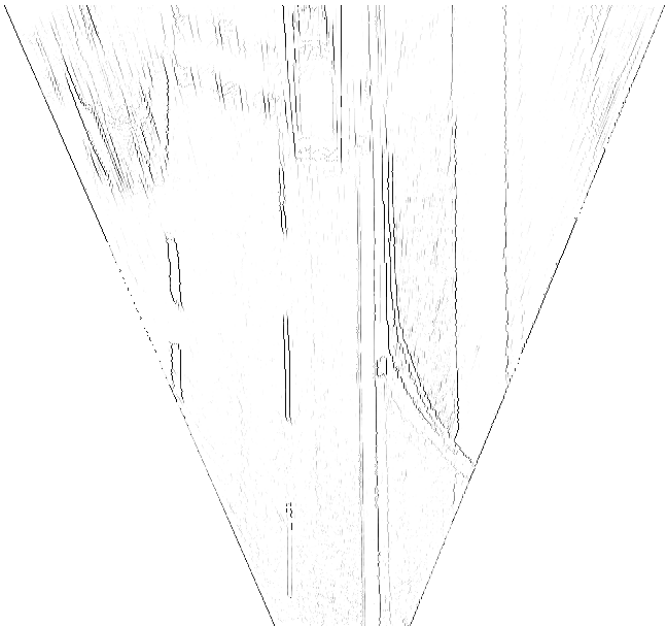


Figure 3.1.: Canny Edge Detection applied to the BEV image shown in figure 2.6b (darker pixels represent higher edge values)

segment is detected, a feature is generated in the middle of that segment. The following code example illustrates the line segment search. Note that a scale of 30 was used for the bird's eye view image, so 1 meter corresponds to 30 pixels in the BEV image, and a minimum line length of 10 pixels corresponds to 33 centimeters:

```
/* scale: 1 m = 30 px */
minLineLength   = 10     // 10 px -> 33 cm
minEdgeValue    = 30     // canny edge image: 0..255
minLinePortion = 0.9     // at least 90% of line elements >= minEdgeValue

foreach pixel [x,y] with Canny edge value >= minEdgeValue
    for minLineLength connected pixels in negative driving direction (y)
        count portion of pixels with edge value >= minEdgeValue
    if portion >= minLinePortion
        add feature at position [x,y+minLineLength/2]
```

Figure 3.2 illustrates the individual steps of the line segment search (for a minimum line length of 6 pixels). The red bordered pixel on the left side is the currently analyzed pixel. The pixels within the minimum line length in y direction (green dotted frame) with a high edge value are labeled with a red check mark. The blue cross on the right side marks the feature which is added at the center of the line segment. Note that in this figure a darker pixel represents a higher edge value and a white pixel represents a value of 0.

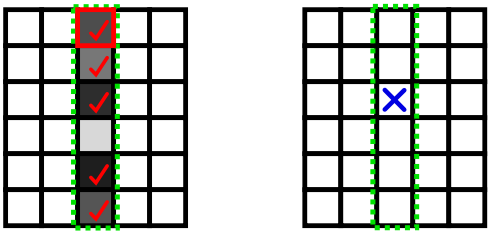


Figure 3.2.: Searching line segment features with a minimum line length of 6 pixels. In this image, a dark pixel represents a high edge value. A feature is added at the center of the line segment if the segment contains enough pixels with high edge values.

Regarding the different characteristics of possible traffic scenes and image quality, this feature detection is performed twice, with two different parameter sets. In addition to the parameters in the code example above, a second feature search finds line segments of the same minimum length, but with a minimum portion of 70% of the pixels having a minimum edge value of 80. This second parameter set detects line segments with higher edge values, but with more interruptions. Furthermore, some *single edge features* are added at locations of just high edge values, not regarding their connectivity to other edge elements. This way, features are detected even in situations with stronger curves. To control the number of features and to achieve a nearly uniform distribution of the features in the image, a minimum distance between the single features can be defined. Figure 3.3 shows the result of the feature detection process in the bird's eye view image of a typical traffic scene. Figure 3.4 shows an example of a curved road, where the line segment search in driving direction would not provide enough important features. Only the additional single edge features lead to a good coverage of features in the image.

3.1.2. Feature Classes

Since the spatial relations have to be learned for different feature types, the features have to be classified first. To obtain a set of features for classification, 5000 image patches from several video sequences of different road scenes were extracted. After the feature detection, as described in the previous section, at every feature location, an image patch of 60x80 pixels around the feature position was extracted and saved. These image patches were later used to extract a feature descriptor (see section 3.2.3). Since the learning process was performed on projected images with a scale of 30 (1 meter $\hat{=}$ 30 pixels), these image patches correspond to a size of 2×2.67 meters in the real world. This way it was ensured that the saved patch would contain enough information to classify the feature. These patches were analyzed, and from the complete feature set, 3574 features were classified manually into 18 classes. The remaining features could not be assigned to

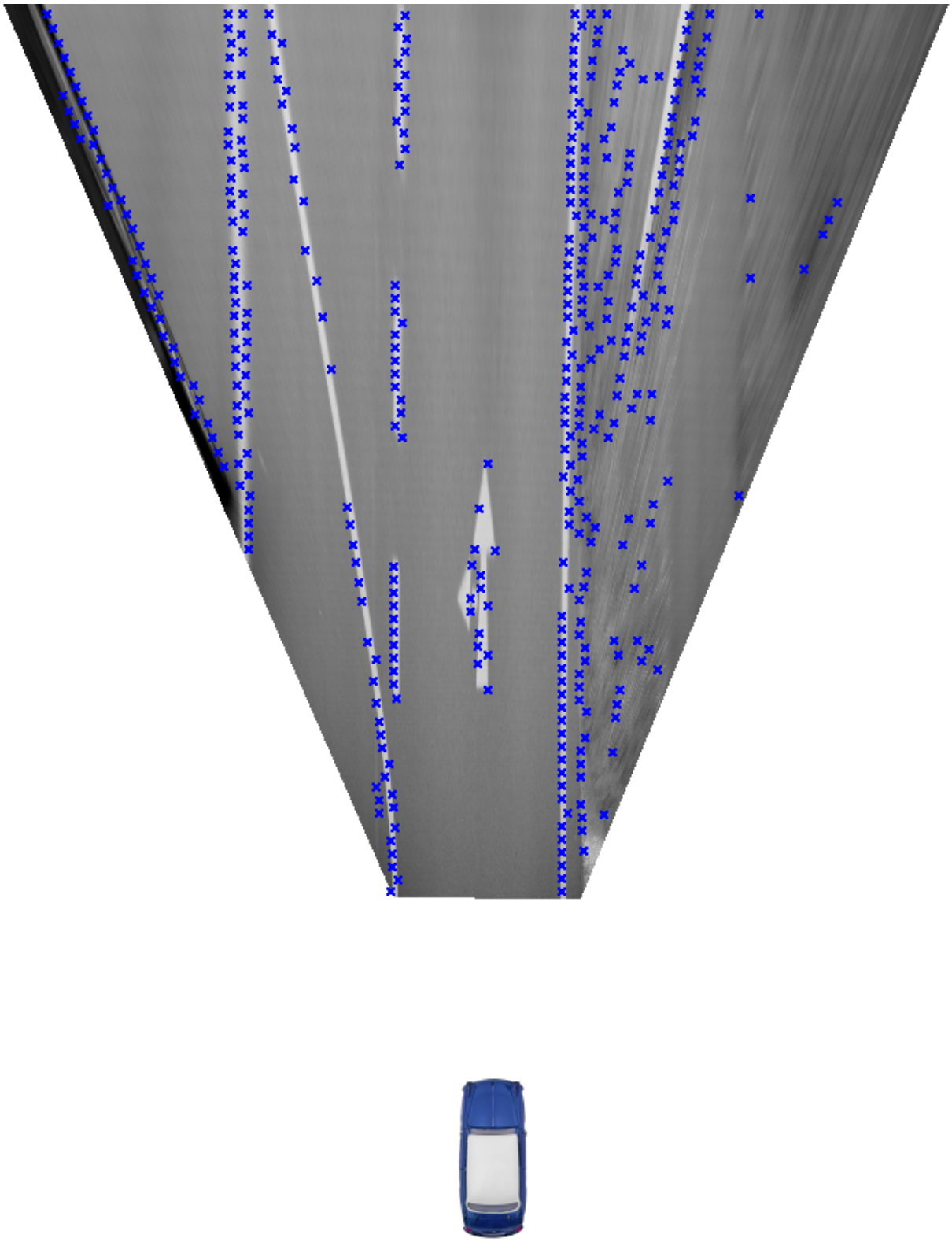


Figure 3.3.: Feature detection with a minimum distance between features of 0.3 meters

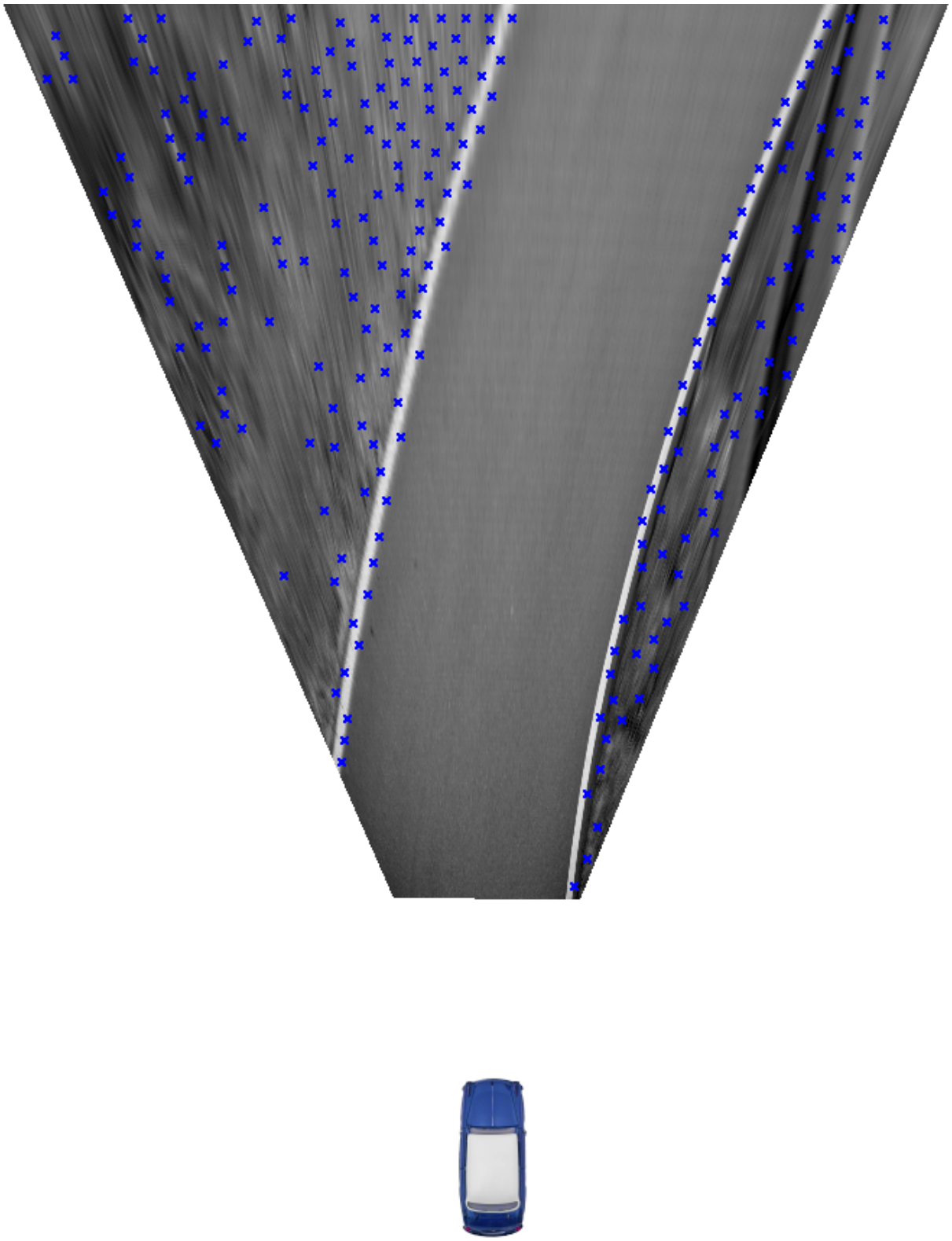


Figure 3.4.: Feature detection in a curved road scenario with a minimum distance between features of 0.5 meters

one class unambiguously. The feature analysis led to the generation of 18 feature classes, which are listed in table 3.1.

Table 3.1.: Feature classes with example image patches

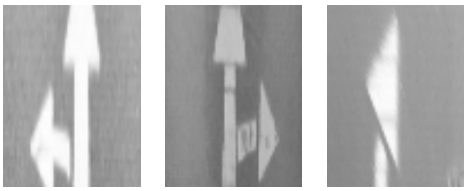
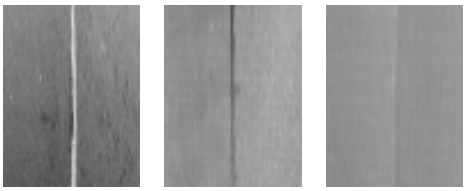
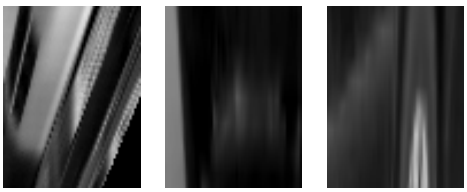
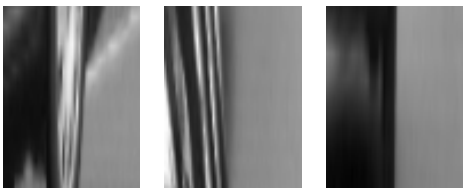
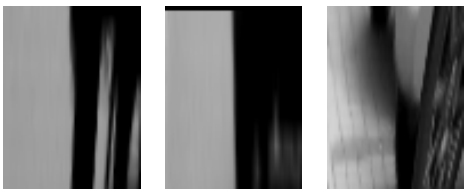
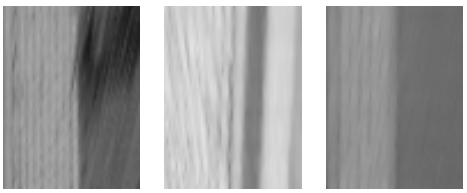
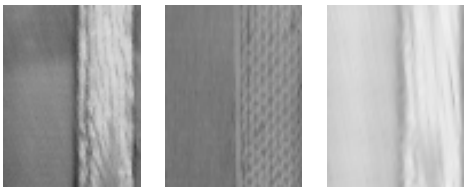
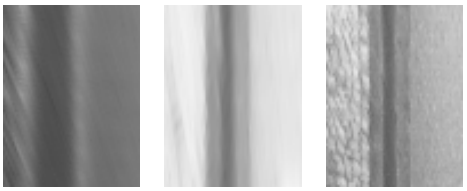
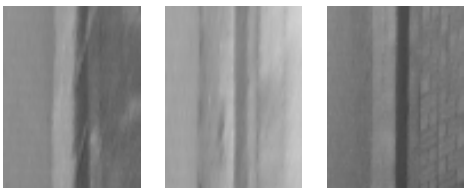
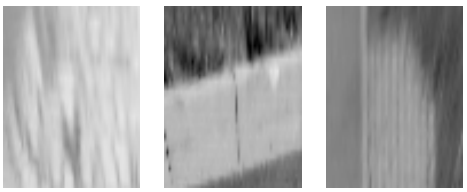
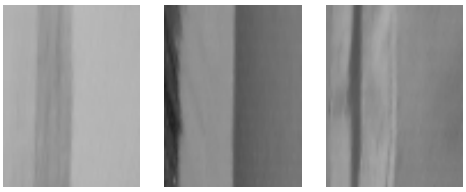
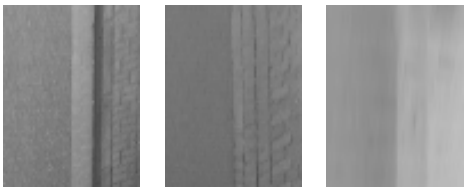
Arrows and Writing	*Asphalt Weld*
	
Car Complete	*Car Left*
	
Car Right	*Cobble Left*
	
Cobble Right	*Curb Left*
	
Curb Right	*Error*
	

Table 3.1.: (continued)

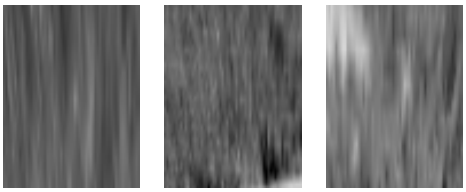
Flat Cobble Left



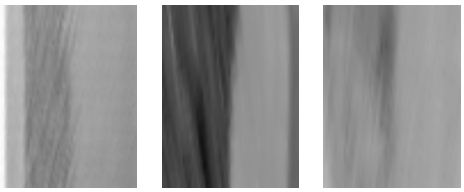
Flat Cobble Right



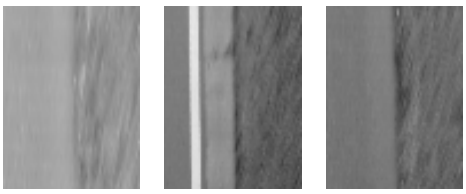
Grass Complete



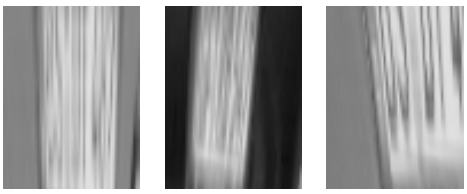
Grass Left



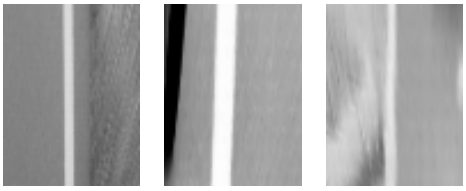
Grass Right



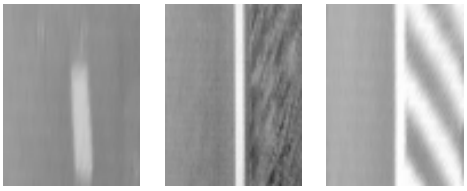
License Plate



Marking Left



Marking Right



The feature position inside the image patch is always the center pixel at coordinates (30,40). If a feature belongs to the feature class *Curb Right*, then the rising edge of the curbstone begins on the right side of the center of the extracted image patch. Of course, the image patch of one feature class can also contain a feature of another class. So if, e.g., a grass area starts on the right side of the center pixel, there might also be flat cobblestone pavement and a lane marking on the left side. Anyway, that feature has to be classified as *Grass Right*.

Most of the feature classes represent locations in the image where a certain structure begins or ends, e.g., *Curb Right* or *Grass Left*. Exceptions are the classes *Car Complete*, *Grass Complete* and *License Plate*, where the detected feature lies inside a certain structure. The class *Error* contains all kinds of features which clearly do not represent one of the other classes. This should not be mistaken for the portion of detected features which

could not clearly be classified. To determine the class of a feature, a descriptor has to be extracted from the image patch, which can be processed by a classifier. Such a feature descriptor can be a vector of values. The descriptors and machine learning techniques used to solve this task of feature classification will be described in the next section.

3.2. Classification

In the previous section, the different types of features used for the task of lane detection were introduced. A set of feature patches was extracted from images of several test sequences and classified manually. This section describes the automatic classification of new features.
There is a wide range of classification applications, especially in computer vision. Since there are many well developed methods for assigning a certain class to a data sample, many tasks can be solved reliably. Examples for such tasks are camera-based quality control, face recognition, or optical character recognition. In these examples, the input data for the classification is derived from the pixel values of an image, and the output is the corresponding class which this data sample is assigned to. While in quality control the two classes *Good* and *Bad* can be enough, the face recognition might distinguish between many different faces. Hence, every face represents a class. Also the task of face detection (without distinguishing between different persons) can already be considered as classification, since an image (or a location inside the image) has to be labeled as *Face* or *No Face*. In optical character recognition, each class represents a character. In these examples, the feature classes have to be defined before, and the classifier needs training samples of each class. When a classifier is trained with already labeled data samples, this procedure is called *supervised learning*. This machine learning method presumes a prior definition of classes.
Another application of machine learning is finding (previously unknown) patterns in data. This way, samples with common characteristics can be grouped, and feature classes can be generated. Since the training data do not have to be labeled with classes previously, this process is called *unsupervised learning*. It is often also referred to as clustering, since in many applications, data samples are grouped into clusters. In fact, clustering is only one area of unsupervised learning.
A different type of learning is the *reinforcement learning*, where a system has to reach a certain goal. The system is rewarded or penalized for trial and error actions, so it finally learns in which state which action most probably leads to the goal.

Although for abstract features, as described in section 3.1, unsupervised learning could be considered, for the classification of the feature classes defined in the previous section, supervised learning is applied. The input data x for the classification is some feature descriptor, which is, e.g., a vector of pixel values extracted from the image patch. The

output $y \in [1..18]$ is a number between 1 and 18 which represents one of the 18 feature classes. The training set contains many of these training pairs (x, y) consisting of a feature and its label. With the training data, a model can be created which describes the relations between input and output. In this context, we distinguish between *generative* and *discriminative* models. A *generative model* is a complete probabilistic model of the input and output data which models the joint probability distribution

$$P(X, Y) = P(Y|X)\, P(X) = P(X|Y)\, P(Y)\,. \tag{3.7}$$

This means that not only the output, but also input data can be *generated* from this model. The *Hidden Markov Model* and *Naive Bayes* are common generative models. *Discriminative models*, on the other hand, only model the conditional probability distribution $P(Y|X)$, so only the output (class) can be sampled for a given input (feature vector). Examples of discriminative models are *Support Vector Machines*, *Decision Trees*, *Logistic Regression* and *Neural Networks*.
For the classification of the features in the vehicle environment, the performance of several classifiers was analyzed, mostly discriminative models. Of course, all these classifiers have different characteristics and are trained differently, but a coarse understanding of how a classifier can be built is described here for the case of *Logistic Regression*, which can be derived from linear regression.

3.2.1. Linear Regression and Classification

Linear Regression is not a classification method, but in some cases, it can serve as a classifier and it can be seen as a basis for a real classifier, which will be described in the next section. Let us consider the n-dimensional input variables $x \in X$ and the (discrete) classes $y \in Y$. For a set of m samples, one input sample $i \in [1..m]$ is referred to as $x^{(i)}$, hence the i-th training pair of input and output is $(x^{(i)}, y^{(i)})$. A function $h(x)$ is needed which maps a sample $x^{(i)}$ to its feature class $y^{(i)}$. In linear regression, this is a linear function

$$h_\beta(x^{(i)}) = \beta^\top x^{(i)} \tag{3.8}$$

with the regression parameters β, and it is called the hypothesis, since for a new input sample, it estimates the class y.

Figure 3.5 shows an example of a 1-dimensional feature space, with two classes $Y = \{0, 1\}$. The hypothesis $h(x)$ is calculated with linear regression. In this example with the univariate input variable x with $n = 1$, the hypothesis is the linear function $h_\beta(x^{(i)}) = \beta_0 + \beta_1 x^{(i)}$. To write this expression in vector form, according to equation 3.8, an additional feature vector element $x_0 = 1$ is inserted, so that $x \in \mathbb{R}^{n+1}$ and $\beta \in \mathbb{R}^{n+1}$.

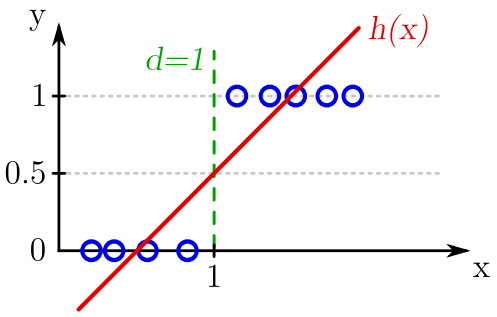


Figure 3.5.: Example of a 1-dimensional feature space and classification into two classes (0 and 1) with linear regression. The green dotted line is the decision boundary $d = 1$.

To determine the parameters β of the hypothesis, a cost function

$$J(\beta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\beta(x^{(i)}) - y^{(i)} \right)^2 \tag{3.9}$$

is defined, where $h_\beta(x^{(i)})$ is the hypothesis for the sample $x^{(i)}$, and $y^{(i)}$ is the real class (the correct output) of the training sample, and m is the number of training samples. The error metric used in this cost function is the squared error (of course, other metrics can be used, too). To find the parameters β with the smallest classification error, the cost function has to be minimized, thus the partial derivatives are set to 0:

$$\beta = \arg\min_{\beta} J(\beta) \quad \Rightarrow \quad \frac{\partial}{\partial \beta_j} J(\beta) \stackrel{!}{=} 0\,, \quad \forall j\,. \tag{3.10}$$

The linear hypothesis function ensures a convex cost function, which has exactly one minimum. So, equation 3.10 is sufficient for the minimization of the cost function. The partial derivatives

$$\frac{\partial}{\partial \beta_j} J(\beta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\beta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} \tag{3.11}$$

can now be used to refine the parameters by gradient descent or to calculate the parameters directly using the normal equation $\beta = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top y$, where $\mathbf{X}$ is an $m \times (n+1)$ matrix (every row contains the feature vector elements of one input sample, starting with $x_0 = 1$ in the first column), and y is the m-dimensional vector of assigned class labels. Gradient descent works well also for many features, but implies the selection of a further parameter α, the step size of the descent at every time step:

$$\beta_j := \beta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\beta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}\,. \tag{3.12}$$

For the normal equation, no additional parameters are needed, but the inverse of the matrix $(\mathbf{X}^\top \mathbf{X})$ has to be calculated, which can get computationally expensive if the number

of features is large ($O(n^3)$ for an $n \times n$ matrix). Returning to the example in figure 3.5, the decision for the estimated class can be made by

$$\hat{y} = \begin{cases} 0 & \text{if } h_\beta(x) < 0.5 \\ 1 & \text{if } h_\beta(x) \geq 0.5 \end{cases} . \tag{3.13}$$

It can be seen in figure 3.5, that the calculated parameters for this hypothesis (the red line in the figure) are $\beta = (\beta_0, \beta_1) = (-0.5, 1.0)$. This shows the close relation between the hypothesis generated by linear regression and the decision boundary (green dotted line), because inserting the parameter values into equation 3.8 for the threshold of $h_\beta(x) = 0.5$ and transforming leads to the decision boundary $d = 1$:

$$h_\beta(x) = \beta_0 + \beta_1 x = 0.5 \quad \Rightarrow \quad x = \frac{0.5 - \beta_0}{\beta_1} = \frac{0.5 - (-0.5)}{1.0} = 1 \,. \tag{3.14}$$

However, in most cases linear regression, which is a common method for fitting a model into a set of data samples, is not a good classifier. Reviewing the example in figure 3.5, two characteristics of the linear regression get obvious. First, the hypothesis can take values far away from the target values of 0 and 1, even negative values are possible. It would be much more useful to have a function which returns values between 0 and 1, allowing an estimation of the reliability of the classification result. Secondly, sporadic training samples with far different values (outliers), will lead to great changes in the hypothesis, and so in the decision boundary. A single training sample belonging to class 1 with a much higher value of x could result in a decision boundary, which would assign all other training samples to class 0. This is where a change in the hypothesis function leads to a great improvement and is a great step to a real classification method, which will be described in the next section.

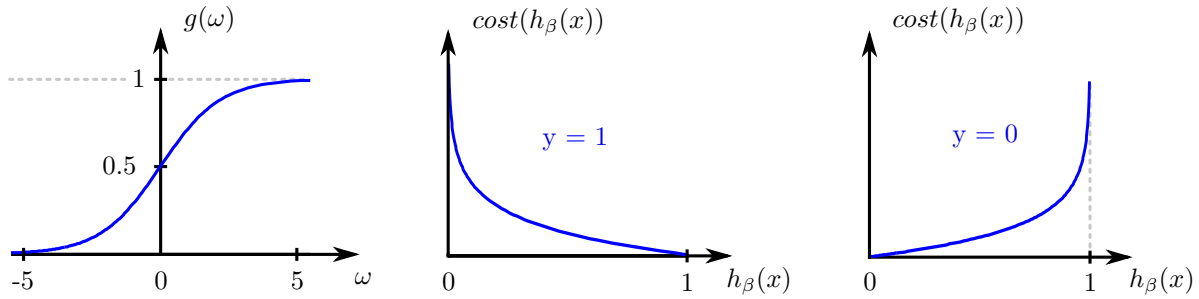
3.2.2. Logistic Regression

In contrast to linear regression, logistic regression is a real classifier. The output function (hypothesis) is the logistic function (also known as sigmoid function)

$$g(\omega) = \frac{1}{1 + \mathrm{e}^{-\omega}} \,, \tag{3.15}$$

which is shown in figure 3.6a. This function has the advantage that for any input x, the output value is between 0 and 1, and the probabilities of a feature to be classified to one class (0) or the other (1) sum up to 1, which means $P(y = 0|x; \beta) + P(y = 1|x; \beta) = 1$. As parameter ω of the logistic function, the hypothesis derived in the previous section is used, so the new hypothesis function is

$$h_\beta(x) = \frac{1}{1 + \mathrm{e}^{-\beta^\top x}} \,. \tag{3.16}$$



(a) The logistic function **(b)** The costs for $y = 1$ **(c)** The costs for $y = 0$

Figure 3.6.: The functions used to create the hypothesis and the cost function in logistic regression

Since this hypothesis is not a linear function, the cost function from equation 3.9 is not convex anymore, which causes problems (local minima) for the gradient descent. For this reason, the costs are defined as

$$cost\Big(h_\beta(x), y\Big) = \begin{cases} -\log\Big(h_\beta(x)\Big) & \text{if } y = 1 \\ -\log\Big(1 - h_\beta(x)\Big) & \text{if } y = 0 \end{cases} . \tag{3.17}$$

Figure 3.6b and 3.6c show the used functions for the costs of $y = 1$, and $y = 0$, respectively. It can be seen that the costs of a false classification are very high. Combining these costs in one equation leads to the convex cost function

$$J(\beta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\Big(h_\beta(x^{(i)})\Big) + (1 - y^{(i)}) \log\Big(1 - h_\beta(x^{(i)})\Big)\right] . \tag{3.18}$$

The derivative of the cost function $J(\beta)$ results to be the same as in linear regression (equation 3.11), the only difference is the new hypothesis function $h_\beta(x)$ (equation 3.16). As described in the previous section (equation 3.12), this term can be used for gradient descent to calculate the parameters β.

This classifier can already decide if a new feature sample belongs to one class or another. For the set of all vehicle environment feature classes $C = \{c_1, ..., c_{18}\}$ from section 3.1.2, this two-class decision can be used to build 18 one-vs.-all classifiers for every feature class c_i. Each classifier then decides between the two classes $Y = \{y_1, y_2\}$ with $y_1 = c_i$ and $y_2 = C \setminus c_i$. Since the hypothesis $h_\beta^{(i)}(x)$ of the i-th one-vs.-all classifier estimates the probability of a feature sample belonging to class c_i, $P(y = c_i|x; \beta)$, the class with the highest probability can be chosen as the final result:

$$y = c_i \quad \text{with} \quad i = \arg\max_i h_\beta^{(i)}(x) . \tag{3.19}$$

In addition to this one-vs.-all method, another possibility for multiclass classification is the one-vs.-one approach. In this case, for p classes, $p\,(p-1)/2$ classifiers (153 for $p = 18$) have to be trained instead of p one-vs.-all classifiers. Although the high number of classifiers suggests a higher computational effort, this is not necessarily the case, but depends on the type of classifier. If the training complexity exceeds $O(n^2)$, with n = number of training samples, which is often the case (e.g., for *Support Vector Machines*), the training of many subset classifiers might be even faster (Milgram et al., 2006). But for calculating an accurate probability measure, further post-processing is needed (Renninger and Malik, 2004).

The logistic regression is only one example of a classifier. Other classifiers are trained in a different way, but all of them process the same input data and provide a certain output. Before comparing the performances of the different tested classifiers, the next section will describe the input data x, because the quality of the results strongly depends on the chosen input vector: the feature descriptor.

3.2.3. The Feature Descriptors

The extracted and manually classified feature patches build the basis of the classifier training data set. To achieve a good classification performance, not only the selection of a suitable classifier is crucial, but also the decision how the input data can be represented best. The feature descriptor defines in which way values are extracted to describe the characteristics of the data. It depends on the chosen feature descriptor if the feature is, e.g., invariant against rotations, translations, scaling, or varying lighting conditions. However, these desired characteristics generally lead to more complex features requiring more effort for feature extraction and further processing. For this reason, in this work, the performance of different types of feature descriptors is analyzed, from very simple descriptors (row of pixel values) to more complex structures (Local Binary Pattern histograms). Figures 3.7 to 3.11 show the structures of the descriptors. The black raster in the figures is a schematic representation of the pixels (not their actual size and number), and the blue cross indicates the feature location.

The feature vector of the first descriptor contains the gray values of a horizontal line of 41 pixels (20 pixels on each side of the feature location), as shown in figure 3.7. Since the image patches were extracted with a scale of 30 pixels per meter, the 41 pixels correspond to 1.36 meters in the real world. This seems to be a large area for the classification of, e.g., a lane marking, but for more complex feature types, e.g., *Curb Left*, it is necessary to consider a wide neighborhood of the feature location (compare table 3.1). With this horizontal line of 41 pixels, two additional feature descriptors were created. Not only the original gray values were used, but also a normalized version of this profile descriptor was

built, where all values lie between 0 and 1, to achieve higher illumination invariance. For another variant of the profile descriptor, invariant to image noise, the corresponding area in the feature patch was median filtered with a 3×3 filter kernel, before extracting the feature vector.

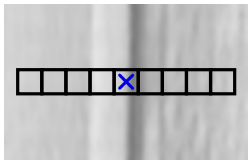


Figure 3.7.: Horizontal gray value profile

Figure 3.8 shows the second feature descriptor type. The pixel values of a rectangular region of 21×31 pixels (21 rows, 31 columns) around the feature location were extracted and written consecutively (row by row) into the feature vector, resulting in a vector of 1271 elements.

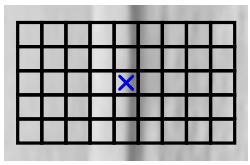
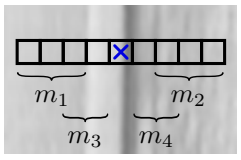


Figure 3.8.: Image patch: rectangular region around the feature location

The third feature descriptor was inspired by the FREAK features (Fast Retina Keypoint (Alahi et al., 2012)), which originally describe features in images by a pairwise comparison of image intensities in regions of different sizes around the feature location. With several intensity pairs, binary strings are created (one pair returns 0 or 1, depending on which intensity is higher). In this work, this approach was used in a modified and simplified way for the 1D case. The horizontal profile from figure 3.7 was divided into several regions of different sizes. A set of 37 pairs of these regions was selected and the corresponding average intensities were subtracted (no binary strings were generated) to build a feature vector of 37 elements. This descriptor is illustrated in figure 3.9 and referred to as FREAK-Inspired Descriptor (FID).



$$f = (m_1 - m_2,\ m_3 - m_4, ...)^\top$$

Figure 3.9.: FID feature with feature vector f

The next descriptor uses *Local Binary Patterns* (LBP) (Ojala et al., 1994). In a rectangular region of 21×41 pixels, for each pixel the 8-neighborhood LBP is computed. Each pixel

of this 8-neighborhood is assigned 0 or 1, depending on whether its intensity is below or above the center pixel value. For all pixels of the defined patch, an 8 bit binary string is created and the histogram of these values is the feature vector with 256 elements (figure 3.10).

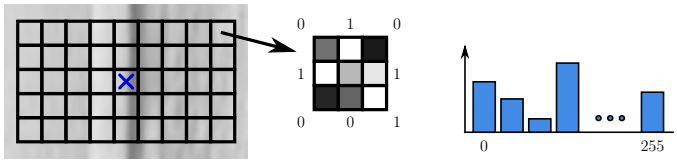


Figure 3.10.: LBP histogram

The last feature descriptor, the LBP$^+$ descriptor, is a combination of the LBP histogram and the horizontal profile, which is shown in figure 3.11. The nature of the feature detection process leads to features which generally separate two different textures (e.g., asphalt and grass), so the LBP Histogram of the whole feature neighborhood does not describe the feature well (see section 3.2.4). For this reason, a descriptor was built which describes the textures on both sides of the feature position. It contains two LBP histograms and also the gray values on a horizontal profile of 55 pixels. The resulting feature vector then contains $256 + 256 + 55 = 567$ elements.

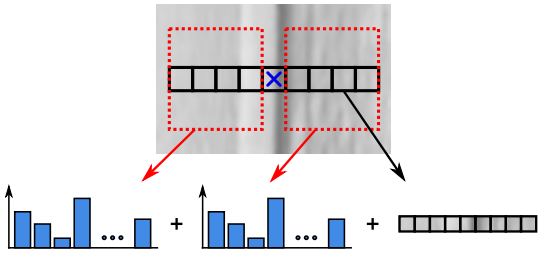


Figure 3.11.: LBP$^+$: Two LBP histograms (from the red-framed areas) + profile

How these descriptors perform with different classifiers will be described in the next section.

3.2.4. Classification Performance

In the previous sections, the principles of a simple classifier (logistic regression) were introduced, and several ways of input data representation (feature descriptors) were presented. There exist many different classifiers, which differ not only in their general classification performance, but also in complexity and runtime in the training and testing phase. Unfortunately, there is not one best classifier. The performance always depends on several properties, e.g., the characteristics of the input data (how can the different classes be

separated?), but the relation between the number of training samples and the feature vector size can influence the performance, too. Furthermore, different tasks might have different demands on the classifier output. Because sometimes it is sufficient to know the most probable class of a feature, but in other cases, also a measure for the classification reliability or even the probabilities of all feature classes are needed. Hence, a classifier has to be found, which is suitable for the given input data and also provides a result that meets the requirements of the given task.

To evaluate the classification performances, 2/3 of the complete data set (randomly chosen) was used to train the classifier and the remaining 1/3 was used as a test set. The percentage of correctly classified samples of the test set serves as a measure for the classification performance. To get more detailed information about the quality of a classifier, ROC curves (Receiver-Operator-Characteristic) can be created, which plot the true positive rate $TP/(TP+FN)$ over the false positive rate $FP/(FP+TN)$ with TP = number of true positives, FP = number of false positives, TN = number of true negatives and FN = number of false negatives. However, to create a ROC curve, the classification has to be repeated several times with different parameters, since each classification process provides one point of the curve, and the area under the curve is a good estimate of the overall classifier performance. However, the portion of correctly classified samples gives a coarse but quick and adequate overview of the performance. Table 3.2 shows the tested classifiers and the classification quality for the different feature descriptors described in section 3.2.3. The tests were performed with the *WEKA DataMining Tool* (Mark Hall, 2009).

Table 3.2.: Classification performance in % of correctly classified samples

Classifier	Profile	Patch	FID	LBP	LBP$^+$
Nearest Neighbor	82.5	78.0	81.2	37.5	82.1
Decision Tree (C4.5)	75.0	74.1	72.8	33.5	72.8
Rule Sets	75.8	71.6	73.6	32.4	73.5
Bayesian Network	63.9	61.9	69.4	25.3	40.9
Multilayer Perceptron	82.0	80.3	77.0	17.9	40.7
Logistic Model Trees	74.7	74.2	67.5	46.4	85.9
SVM (polynomial kernel)	80.6	78.2	74.5	50.5	89.1

In the following, the classifiers used in the experiments are described briefly.
The **Nearest Neighbor** classifier (Aha et al., 1991) compares a new input sample with the training data and finds the nearest sample of the training set in the feature space. The class of this nearest neighbor is then assigned to the new data sample. A simple extension is to analyze the k nearest neighbors and choose the class of the majority of the k samples.

The **C4.5** algorithm (Quinlan, 1993) is a popular method for constructing **decision trees**. A decision tree for classification (also *classification tree*) is usually a binary tree where the leaves represent the classes. Starting at the root, the nodes (except the leaves) represent rules for continuing the way either to the left or the right child, until one of the leaves - and so the decision for a class - is reached. Each of these rules can be considered as a decision boundary in the feature space (compare figure 3.5) and every node represents a subspace of the feature space.
The **Rule Sets** classifier (Frank and Witten, 1998) is closely related to the C4.5 algorithm, because it builds several (partial) decision trees and derives a set of rules for the classification. The use of these partial decision trees avoids the tedious global optimization of complete decision trees. This difference is supposed to show its benefits especially in the case of noisy training data.
A **Bayesian Network** is a probabilistic graphical model which describes dependencies between random variables with a directed acyclic graph. The nodes represent the random variables and the edges represent their conditional dependencies. The learning of this generative model includes creating the graph structure and learning the related parameters.
The **Multilayer Perceptron** is a neural network with multiple layers of nodes. A neural network is a directed graph, where each node of one layer is connected with a certain weight to all nodes of the following layer. The first layer is the input layer and the last is the output layer. In the training phase, where the output is known, the weights of the network are modified by backpropagation to minimize the error of the overall mapping of the input to the output.
Logistic Model Trees (Landwehr et al., 2005) are decision trees which contain logistic regression functions at their leaves (compare section 3.2.2). The advantage over classical decision trees is that the Logistic Model Trees are generally smaller, because the last branches are substituted by regression functions. They do not tend so much to overfitting, because at a certain level of the tree, there might be only a small number of samples left in a branch, so a further splitting is not recommendable, but a logistic function models the remaining data better (Perlich et al., 2003).
Support Vector Machines insert a hyperplane as decision boundary into the feature space. The distance between this decision boundary and the closest training samples is maximized, and these closest samples are the support vectors. The hyperplane can only separate samples linearly. In the case of not linearly separable data, the feature space is transformed to a space of higher dimensionality where the samples can then be separated linearly. This transformation is done by a kernel function, and this method is called the *kernel method* or *kernel trick*. The maximization of the distance between decision boundary and training data ensures good classification performance even if the test samples are not represented perfectly by the training data.

The classification results in table 3.2 help to choose a feature descriptor and a suitable classifier. The simple (and computationally efficient) *Profile* descriptor already allows good classification results. Experiments with a normalization or median filtering of the feature

vector (not listed in the table) did not lead to any improvements. Contrarily, the C4.5 algorithm (decision tree) with the normalized data resulted in 73.0 % and with the median filtered data in 74.5 % correctly classified samples. This modification of the raw data seems to result in a slightly worse performance (although not statistically relevant). This confirms the assumption, that possible advantages of a higher invariance (e.g., against different lighting conditions or image noise) are compensated by the information loss due to further processing.
The use of more pixels in the feature neighborhood by the *Patch* descriptor does not improve the classification quality, either. This can be due to the fact that the single pixel values in two almost similar patches (of the same class) still differ too much, so, compared to the *Profile* descriptor, the *Patch* descriptor does not add more information, but mostly noise.
The *FID* descriptor was built, since the original (2D) version (Alahi et al., 2012) provided impressive results. The poor classification quality here is apparently caused by the extensive simplification of the approach. In the original version, the choice of the region pairs is supported by machine learning, so the optimal combinations are used.
While *Local Binary Pattern* histograms have already been used successfully for texture classification, also in vehicle environment perception (Seibert et al., 2013), they are not suitable in this case, as already mentioned in the descriptor introduction in the previous section. The features here are detected at edges which typically separate two different textures. To mix these two sides of the feature in one LBP histogram is not reasonable. The classification performance in table 3.2 confirms this assumption.
The LBP^+ descriptor, on the other hand, solves this problem by describing the two sides of the feature separately. The additional gray value profile, which is added to the feature vector, also considers the exact gray values of the transition between the two sides. In addition to the results in table 3.2, it was also tested how the double LBP histogram descriptor (LBP^+) without the gray value profile performs (71.3 % with Logistic Model Trees), and also the normal (single) LBP descriptor in conjunction with a gray value profile (80.7 %). These tests show that the separation into two independent LBP areas, as well as the additional use of the gray value profile, lead to increased performance.
Altogether, for a good classification performance, the LBP^+ descriptor is a good choice. However, if the given task requires higher computational efficiency and allows lower classification reliability, the *Profile* descriptor is an adequate alternative, too.

Within the set of feature classes, the features representing lane markings, *Marking Left* and *Marking Right*, play a special role. Since lane markings appear in many traffic situations and usually cause clear edges in the camera image, features of these classes are detected more frequently than other features. This applies both to the training set and the subsequent classification. The two classes are strongly overrepresented, constituting 45.5 % of the whole training set. Although this is not ideal for classification, it is a realistic reflection of the feature distribution in vehicle environments. Since the classification of features is important, especially in the cases, where no lane markings exist, the classifi-

cation performance was also tested without these two classes, because lane markings are usually easier to classify than other feature classes and can distort the overall result. After removing the classes *Marking Left* and *Marking Right* from the feature set, the Logistic Model Trees with the LBP^+ descriptor still classified 82.1 % of the samples correctly. Only using these two classes and removing all other features, 98.4 % of the samples were classified correctly.

To retrieve a more precise intuition about the classifier performance, instead of applying the test split (2/3 for classifier training and 1/3 for testing), an s-fold cross-validation can be performed. The m training samples are divided randomly into s subsets of the size m/s and the classifier is trained s times with one subset for testing and the remaining samples for training. The average portion of correctly classified samples gives a better overview of the performance than a single test split. A 10-fold cross-validation for the LBP^+ descriptor and Logistic Model Trees lead to 87.0% correctly classified samples.

As described in section 2.3.1, the classified features are used to construct the distribution map. If the reliability of the classification result is available, this information can be used as a weight for the entry in the distribution map. Thus, a classifier should be used, which provides probabilities for the class memberships of the samples. Although table 3.2 shows that the Logistic Model Trees do not guarantee the best (but still a very good) classification result, for every sample x they provide the probability $P(\hat{y} = c_i|x;)$ for each class c_i with

$$0 \leq P(\hat{y} = c_i|x) \leq 1 \quad \text{and} \quad \sum_{c_i \in C} P(\hat{y} = c_i|x) = 1\,. \tag{3.20}$$

This is a big advantage over the Nearest Neighbor, Decision Tree, Rule Sets and SVM classifiers. Furthermore, both the training and the classification of new samples are very fast (contrary to, e.g., the Multilayer Perceptron). Figures 3.12 to 3.15 show results of the classification with the LBP^+ descriptor and Logistic Model Trees.
Since some of the feature classes might not provide important information for the trajectory estimation, it can be reasonable to exclude some of them from the distribution map generation process. Figure 3.16 shows a classification result and the created distribution map, where the classes *Arrows and Writing, Asphalt Weld, Car Complete, Car Left, Car Right, Grass Complete,* and *License Plate* were excluded. How to determine which feature classes should be excluded, is discussed in chapter 6. To enhance the classification performance, a two-stage classification was applied for figure 3.16. The features were classified with the *Profile* and the LBP^+ descriptor and assigned to the class with the maximum sum of both classification results. With the classified features and the previously learned spatial relations, it is possible to estimate the best matching vehicle trajectory.

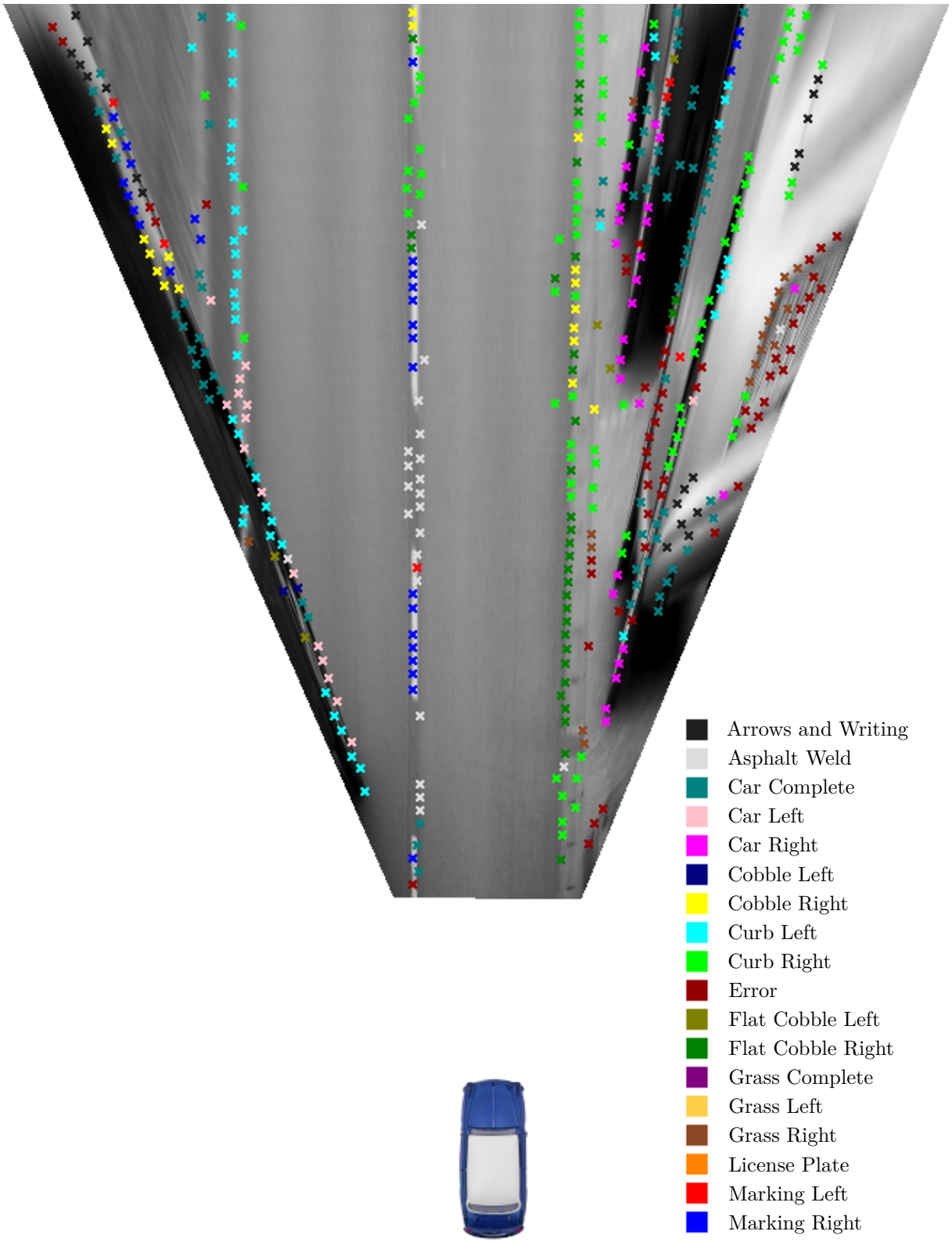


Figure 3.12.: Results of the feature classification (LBP^+ and Logistic Model Trees)

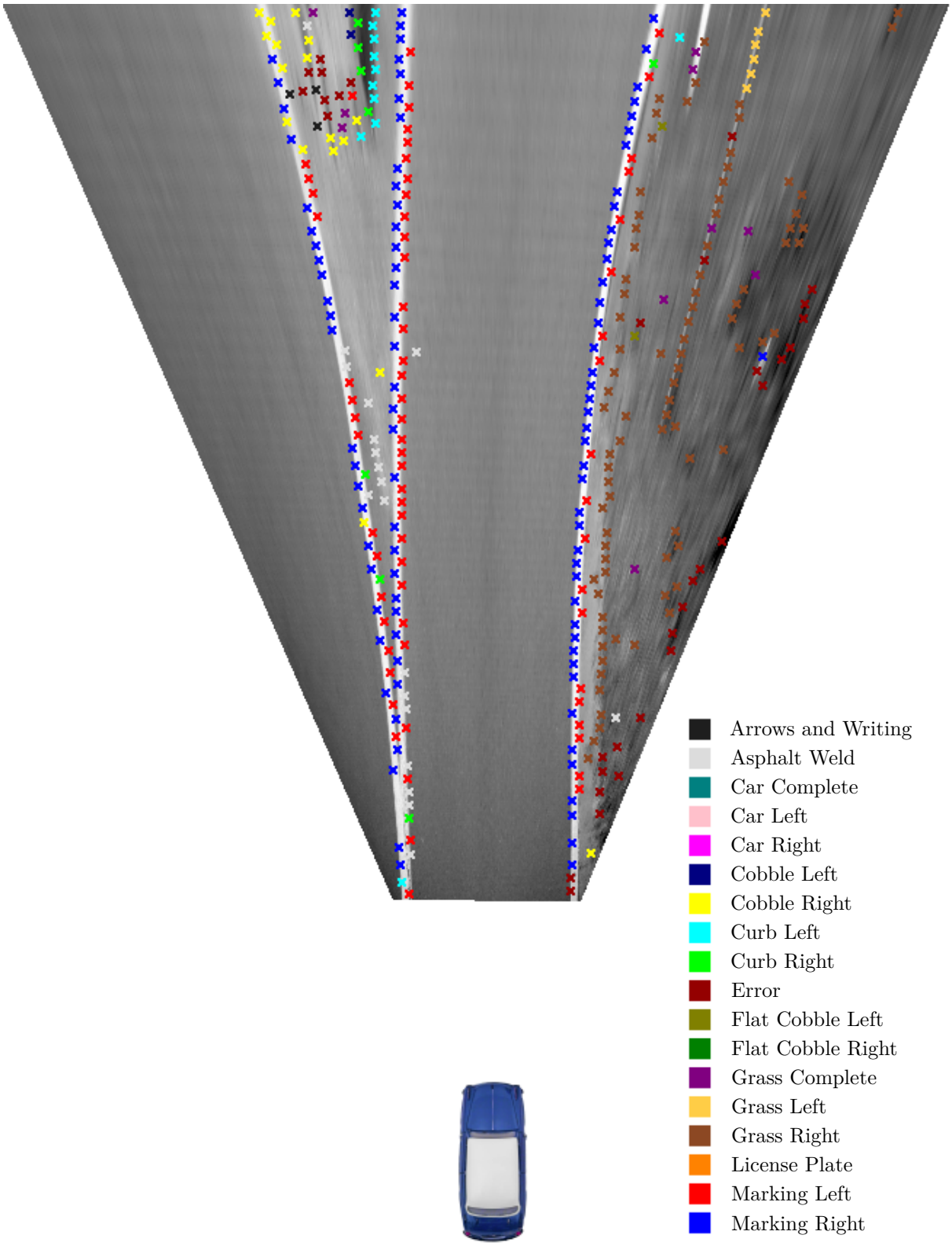


Figure 3.13.: Results of the feature classification (LBP^+ and Logistic Model Trees)

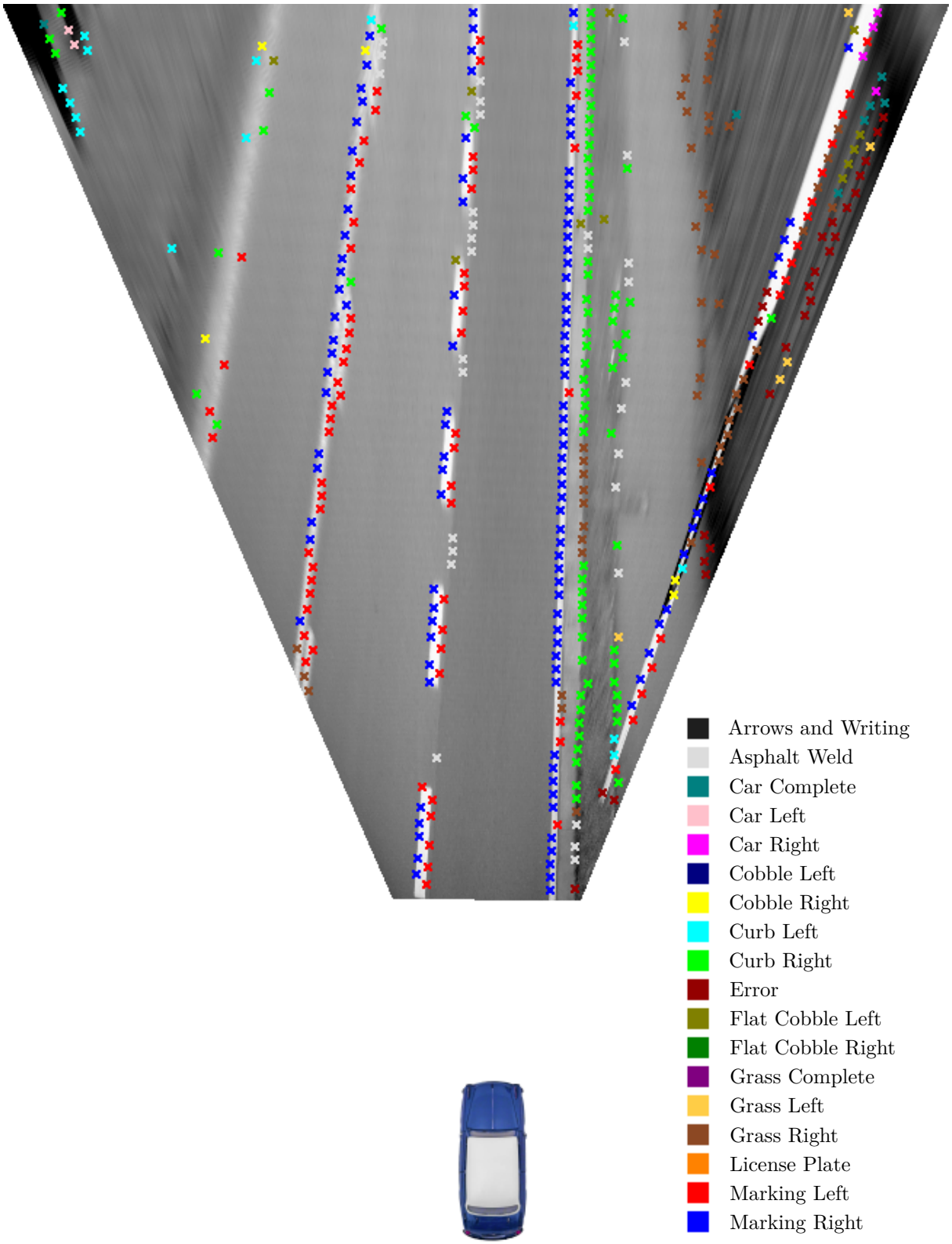


Figure 3.14.: Results of the feature classification (LBP^+ and Logistic Model Trees)

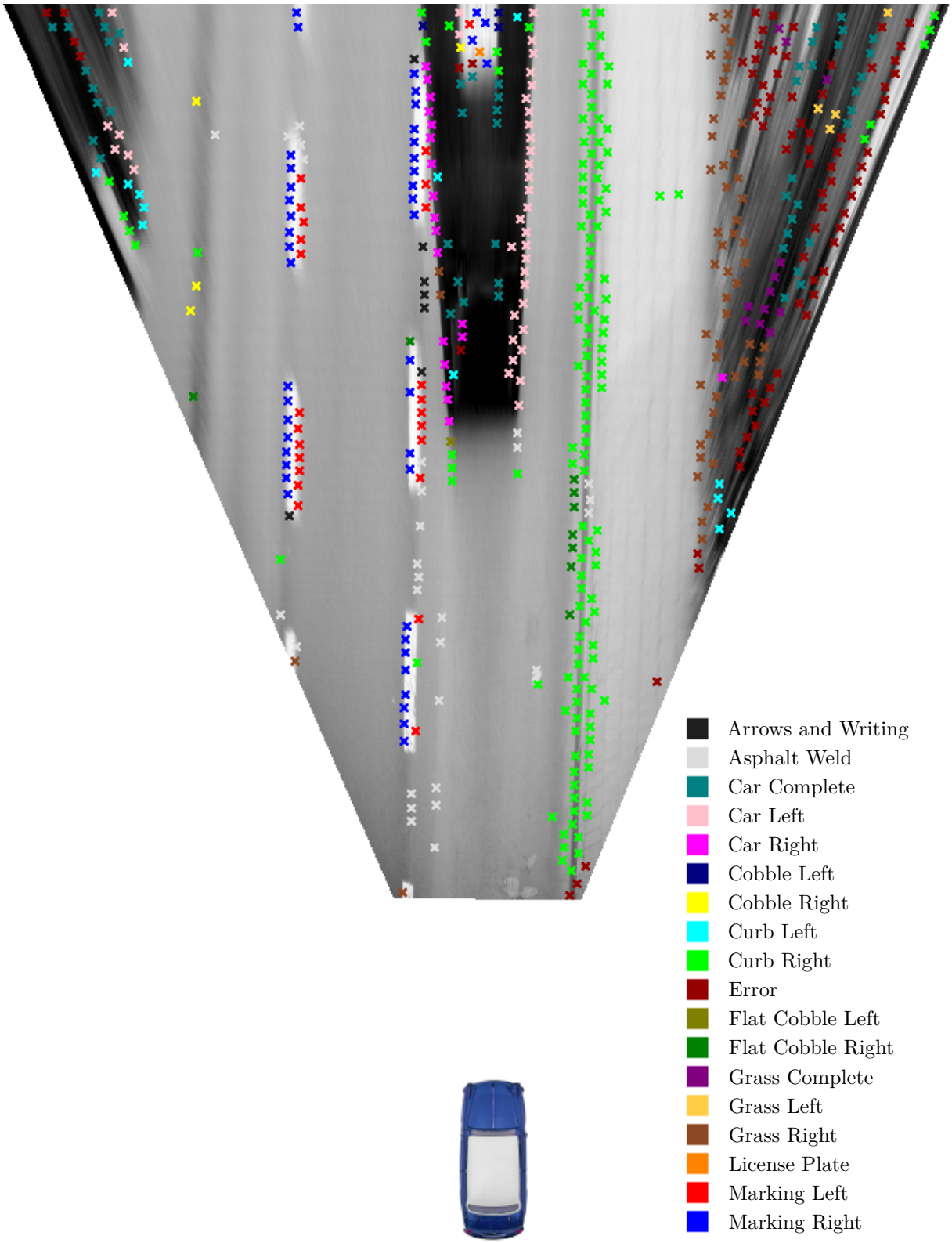


Figure 3.15.: Results of the feature classification (LBP^+ and Logistic Model Trees)

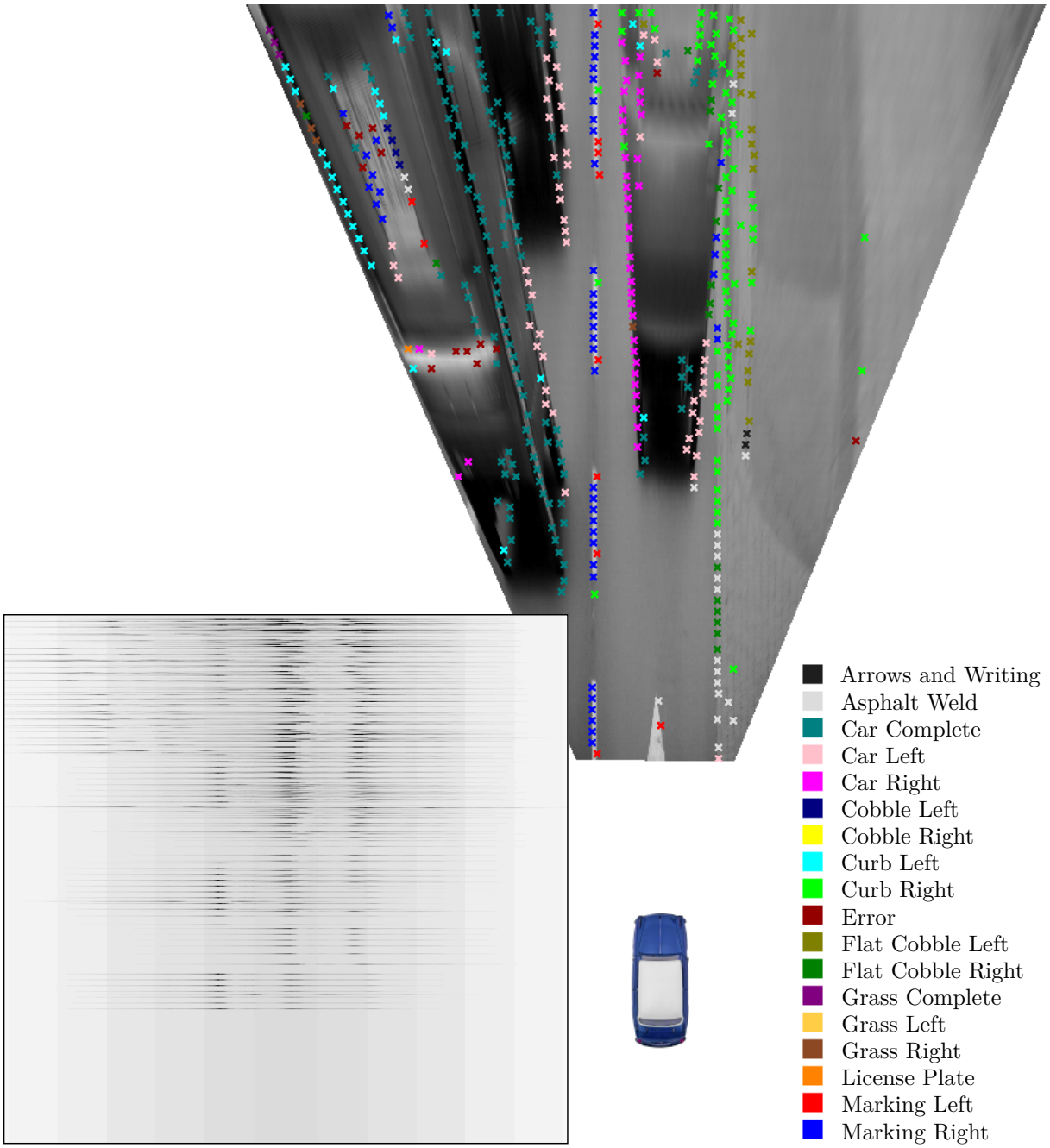


Figure 3.16.: Results of the two-stage feature classification (*Profile+LBP*$^+$ and Logistic Model Trees) and the created distribution map. Excluded classes: *Arrows and Writing, Asphalt Weld, Car Complete, Car Left, Car Right, Grass Complete,* and *License Plate.*

Chapter 4

Model Matching

The previous chapters described how recorded camera images with information about the motion of the vehicle can be used to learn the spatial relations between the vehicle trajectory and local feature points of different classes. With these spatial relations and the detected and classified local features, the distribution map is created. This chapter shows how the distribution map can be used to estimate the vehicle trajectory.

4.1. Modeling Traffic Scenes

For the task of lane detection, there are different possibilities to model the vehicle environment and the components of the traffic scene. Which components have to be included in the model, depends on the task and the method. If only the lane border or the lane markings have to be detected in a given camera image, they can be modeled, e.g., with splines, polynomials, lines or clothoids, and the locations of these curves are defined in the vehicle coordinate system or a given world or map coordinate system. If obstacles are detected (usually with the help of sensors providing depth information, e.g., radar or stereo camera), it is also possible to just save their location. But, since most sensors provide point clouds, to obtain a location an object has to be matched into the points. Another possibility is the use of *Occupancy Grid Maps* (Moravec and Elfes, 1985). Such a map represents a certain area by dividing it into cells of a predefined size, and each cell is (in the simplest variant) assigned to be free space or occupied by an obstacle. So, the process of obstacle detection is reduced to assigning the property *occupied* to all cells containing points from the depth sensors and the property *free* to all cells between the sensor and the first detected obstacle. A variant is the use of a third property *unknown*, assigned to all cells, which have not been observed yet or are occluded by obstacles. Usually, instead of these discrete properties, the cells contain the probability of being occupied by an

obstacle, and the discrete properties can be derived with thresholds. These *Probabilistic Grid Maps* can not only consider uncertainties of a measurement, they also allow a temporal filtering by including the previous cell contents into the probability update process. This characteristic makes the map more robust against sporadic wrong measurements and occlusions, since the probability of a cell is changing slowly over time (with appropriate parameters). While a probabilistic grid map can be used for autonomously driving a given trajectory with obstacle avoidance, it does not provide enough information for trajectory or lane estimation in complex traffic scenarios.

On the other hand, the distribution map, created with local features and previously learned spatial relations, models the vehicle environment considering the influences of any objects on the vehicle trajectory. For a classified feature, both the presence and the absence of the vehicle trajectory is modeled in the lateral offset histogram (compare section 2.2.3). And not only the spatial relations of the local features, but also other available information (e.g., about the expected direction or obstacles), in the form of a probability distribution or a potential field, can be included, as described in section 2.3.2. When a distribution map has been created, it can be used to estimate a vehicle trajectory. The following section describes three different approaches for solving this task.

4.2. Retrieval of the Vehicle Trajectory from Learned Spatial Relations

Taking a look at the distribution map in figure 3.16, the course of the vehicle trajectory is indicated by several local maxima from the lateral offset histograms of the detected features. Additionally, the prior for a forward driving direction increases the values in the center of all rows of the map. So, from the center of the bottom row, which represents the position of the vehicle, a path has to be found to the top row of the map, maximizing the underlying map values, but also complying with certain geometrical constraints.

The next sections will describe three different approaches for fitting a trajectory model into the distribution map, each with highly different geometrical characteristics. The first method is a RANSAC-based (Fischler and Bolles, 1981) line matching, the second is a path search based on dynamic programming, and the third uses predefined curve templates.

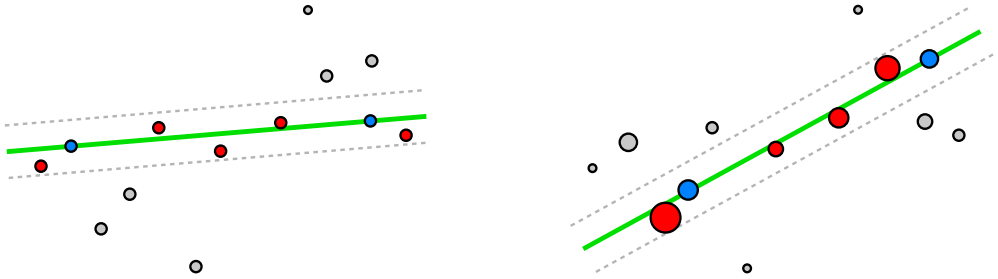
4.2.1. Random Line Search

A very popular and efficient method for model fitting is the *Random Sample Consensus* (RANSAC), presented by Fischler and Bolles (Fischler and Bolles, 1981). In the case of

line fitting, this method randomly selects two samples from a set of data points (since two points define a line), and counts the samples which support this hypothesis (lying close enough to the line). This is repeated several times, and the hypothesis with the highest number of supporting samples (*inliers*) is chosen as the resulting line. Since the distribution map is not a set of samples, some pre-processing is necessary. The distribution map consists of pixels with different values. High values represent a higher probability of the vehicle trajectory to be located at the related pixel. To apply the RANSAC line fitting method, the distribution map is converted into a set of samples. For performance reasons, the number of samples is reduced by dividing the map into cells (e.g., of 10×10 pixels), and each cell is assigned a value, which is the sum of all distribution map pixel values in this cell. If, during the learning of the spatial relations, the lateral offset histogram is constructed with a resolution of 20 bins per meter, and one pixel in the distribution map corresponds to one histogram entry, then a cell size of 10×10 pixels corresponds to 50×50 centimeters. These cells are clusters of the original map values. From all clusters, the 20% with the highest values are chosen as data samples and used for the RANSAC line fitting. For a distribution map of 600×600 pixels (30×30 meters with 20 pixels per meter), the 360.000 pixels are reduced to 720 clusters for the model matching. Since the clusters are not equal data samples, but have certain weights according to the values in the represented distribution map area, instead of the conventional RANSAC, the *Weighted RANSAC* (Schomerus et al., 2014) is used accounting for the weights of the individual samples.

The *Weighted RANSAC* (Schomerus et al., 2014) does not choose the hypothesis with the highest number of inliers, but the one with the highest sum of inlier weights. For every hypothesis, the inliers are not counted, but their weights are accumulated, and the highest weight sum defines the resulting line. Figure 4.1 shows the difference between the conventional RANSAC (figure 4.1a) and the *Weighted RANSAC*. Figure 4.1b illustrates that the *Weighted RANSAC* chooses the hypothesis with the highest sum of sample weights, although it has not the highest number of inliers.

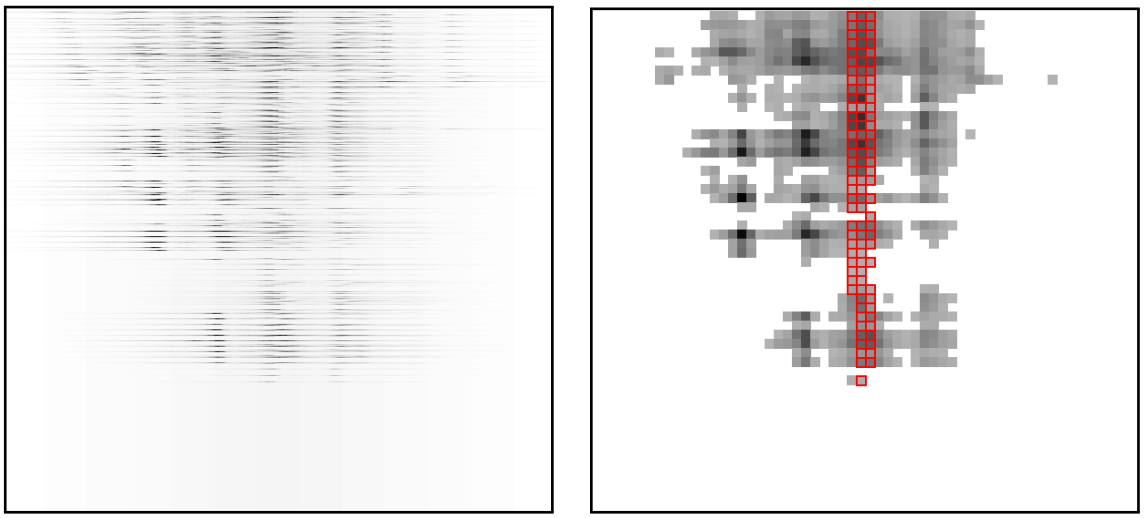
Since for the trajectory estimation, the line should begin at the location of the vehicle, only the second point is selected randomly. Figure 4.2a shows a distribution map and figure 4.2b shows the 20% of clusters with the highest weights (darker clusters have higher weights). The red boxes represent the inliers of the *Weighted RANSAC*. The inlier barrier (allowed distance from the line hypothesis) in this example is set to 60 centimeters. The estimated trajectory by the line model matching is shown in figure 4.3.



(a) RANSAC (Fischler and Bolles, 1981)

(b) Weighted RANSAC (Schomerus et al., 2014)

Figure 4.1.: Conventional and Weighted RANSAC: the green line is the hypothesis, defined by the randomly chosen blue points. The gray dashed lines define the area of inliers (red) supporting this hypothesis. The size of the samples represents their weights.



(a) Distribution map

(b) Clusters and inliers (red)

Figure 4.2.: Line matching: the distribution map, constucted with the lateral offset histograms of all local features of the currend frame (a), is divided into cells representing clusters of the distribution map values (b). These clusters are used for line fitting with *Weighted RANSAC*. The inliers are marked by red boxes.

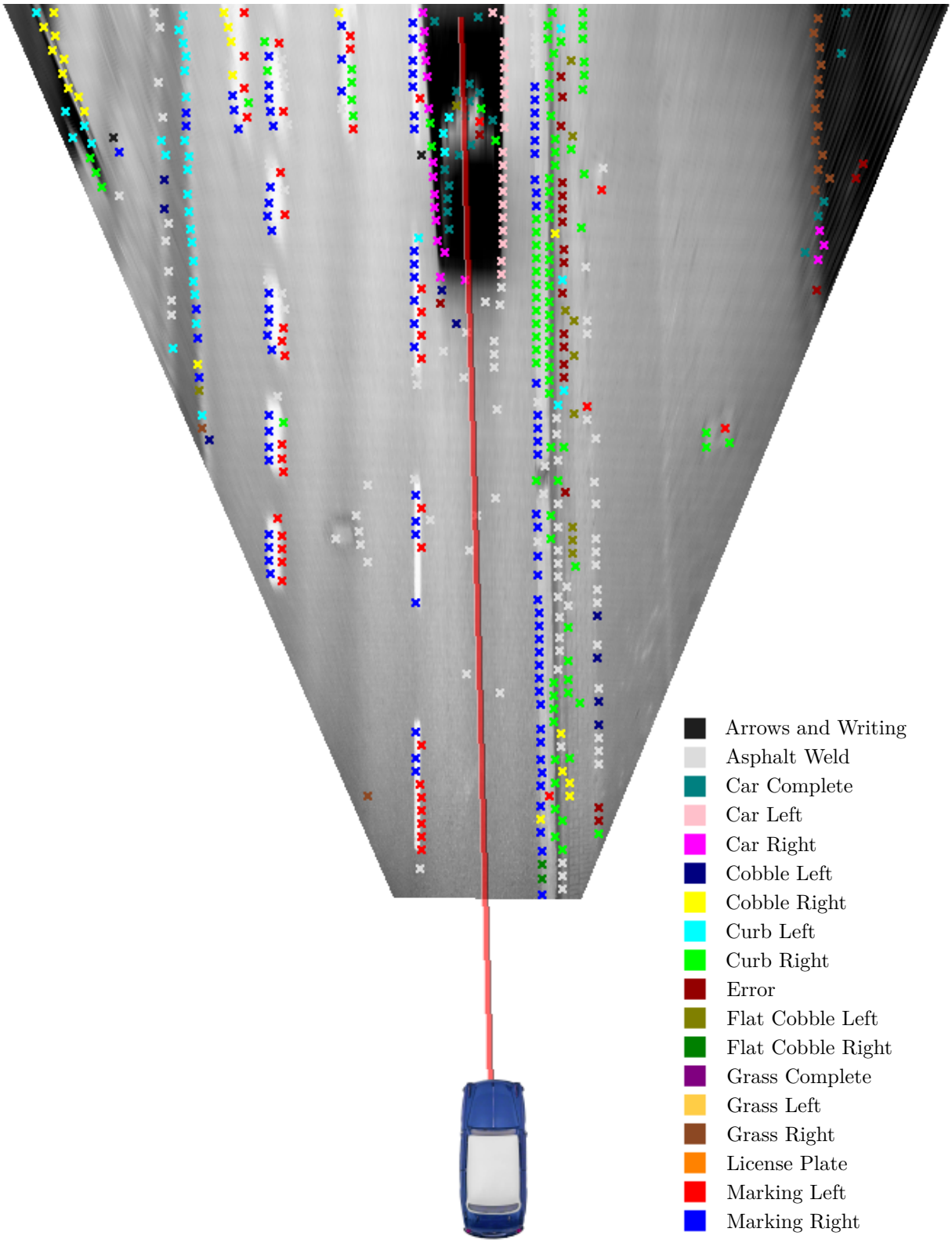


Figure 4.3.: Result of the line matching algorithm

4.2.2. Dynamic Programming

While the first trajectory estimation approach (section 4.2.1) uses a geometrically very restricted model, a straight line, the second matching method allows geometrically much more flexible paths. In this method, the path is only restricted to a maximum angle of 45° to the driving direction, which is sufficient, regarding the traffic scenes and the angle of view of the camera in the provided dataset. The idea is to start the path in the last row of the distribution map at the location of the vehicle and continue upwards, always selecting the best pixel in the next row which belongs to the direct neighborhood of the current path location. That means that for every step to another row (decreasing y) the x-coordinate can at most change by 1. The question is, which of the three possible pixels is the best? A higher distribution map value seems to be more attractive, but another crucial property is, which of these pixels is followed by even better values. So, the sum of the candidate's value and the value of its own best follower is more significant. Since the followers of the followers also have to be considered, this leads to a recursive problem, where the attractiveness of a location in the distribution map is defined as

$$\xi(x,y) = \begin{cases} \Psi(x,y) & \text{if } y = 0 \\ \Psi(x,y) + max\Big(\xi(x{-}1,y{-}1), \xi(x,y{-}1), \xi(x{+}1,y{-}1)\Big) & \text{if } y > 0 \end{cases}, \quad (4.1)$$

where $\Psi(x,y)$, as before, is the distribution map. In this recursive procedure, the solution of the problem is composed of the solutions of subproblems. And since one subsolution contributes to several higher-level solutions, these subsolutions have to be calculated several times. To prevent the multiple calculation of subsolutions and increase the efficiency of the algorithm, the task can be solved by dynamic programming (Bellman, 1954). Instead of recursively calculating the solution, all necessary subsolutions are calculated once iteratively. The dynamic programming approach starts almost at the end of the path, in the second row counting from the top of the distribution map, and calculates the attractiveness $\xi(x,y)$ of every pixel of this row by adding the maximum value of the neighbors in the upper row to its own value. Proceeding until the last row leads to the complete attractiveness map $\xi(x,y)$ without multiple calculations of subsolutions. Now a start point for the path in the last row is chosen (the maximum or a defined location, e.g., of the vehicle), and the path is continued to the top by adding the neighboring pixel of the upper row with the maximum attractiveness value to the path. Table 4.1 illustrates this procedure.

The resulting path is the optimum solution, which means that this method maximizes the sum of attractiveness values under the path (under the given restrictions). Possible variations are to use more than the three direct neighbors of the upper row, or to define

Table 4.1.: Calculating the optimum path with dynamic programming

1. **Distribution map values:**
   The numbers in the cells correspond to values of the distribution map pixels $\Psi(x, y)$, which are the basis for the calculation of the attractiveness $\xi(x, y)$.

0	2	3	1	0	1
1	0	2	1	1	3
0	0	1	3	2	1

2. **Add previous maxima (top-down):**
   The maximum of the neighboring pixels of the previous (upper) row (green box) is added to the current value (blue box).

0	2	3	1	0	1
3	3	5	4	2	4
3	5	6	8	6	5

□ = □ + max{ }

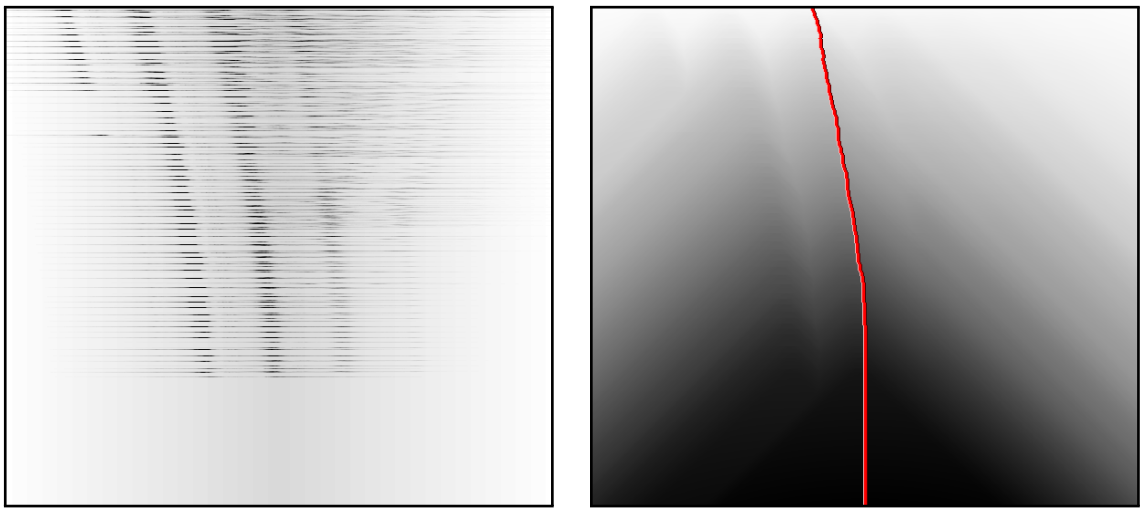
3. **Find maximum path (bottom-up):**
   From the maximum of the lower row, a path (red boxes) to the top is found by always choosing the maximum neighbor in the next (upper) row.

0	2	3	1	0	1
3	3	5	4	2	4
3	5	6	8	6	5

weights for the different directions in the path finding step, e.g., to privilege a certain direction. Figure 4.4 shows the created attractiveness map for a given distribution map.

With this method, very flexible curves can be modeled. This can be a great advantage, but it can also cause problems. The estimated path might contain spontaneous direction changes which are too flexible to represent a vehicle trajectory, even if not all path points are used, but only a certain number of supporting points (e.g., two points per meter). Figure 4.6 shows the result of the dynamic programming approach without further processing of the estimated path.
The result can be improved by a path smoothing which reduces the angle changes between the line segments connecting the path points. Generally, a path defined by arbitrarily distributed single points can be smoothed by shifting each point into the geometric center of the triangle defined by this point and its direct neighbors. Since the algorithm described above provides path points equidistant in driving direction (y-axis of the map image), also after path point reduction (remove all but, e.g., every tenth point), the path can be seen as a discrete function of y, which means $x = p(y)$. In this case, a simple way for path smoothing is a low-pass filtering by a convolution with a rectangular function

$$rect(y) = \begin{cases} \frac{1}{n} & \text{if } |y| \leq \frac{n-1}{2} \\ 0 & \text{else} \end{cases} , \tag{4.2}$$



(a) Distribution map **(b)** Attractiveness map with path (red)

Figure 4.4.: Distribution map (a) and the corresponding attractiveness map (b). As before, darker pixels represent higher values. The calculated path is drawn in red.

where n is the odd number of path points for averaging. The resulting smoothed path is then determined by the discrete convolution

$$p'(y) = \Big(p * rect\Big)(y) = \sum_{i=-1}^{1} p(y-i)\, rect(i)\ . \tag{4.3}$$

For $n = 3$, every path point is shifted to the average of its two direct neighbors and its own value. That means that it is moved exactly into the geometric center of the triangle defined by these points, as suggested above, because the vertical median m of the triangle, which connects the concerned vertex with the midpoint of the opposing side, is divided by the geometric center at the ratio of 2 to 1. So, the concerned vertex contributes to 1/3 to the new path point position and the neighboring vertices contribute the remaining 2/3, which corresponds to the averaging with equation 4.3.

Figure 4.5 shows the rectangular function for $n = 3$ and the effect of the convolution with the original path points as a function of y. The smoothed trajectory corresponding to the distribution map in figure 4.4a is shown in figure 4.7.

4.2.3. Curve Template Matching

The third model matching approach uses predefined curve templates which are compared to the distribution map. This method was inspired by the RALPH algorithm (*Rapidly*

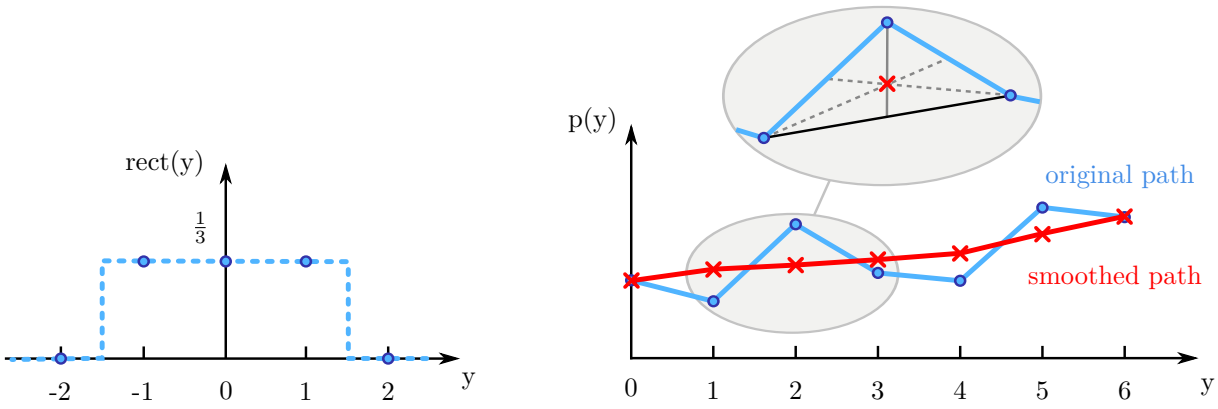


Figure 4.5.: Path smoothing: the rectangular function $rect(y)$ (left) and the result of the convolution (right). Each point of the original path (blue) is shifted into the geometric center of the triangle built with the neighboring points, as shown in the zoom view. The red line is the resulting smoothed path.

Adapting Lateral Position Handler), proposed by Dean Pomerleau (Pomerleau, 1995). RALPH tests several curvatures by shifting the rows of the bird's eye view image by different offsets. The rows corresponding to farther distances from the vehicle generally receive a greater lateral offset. So, each shifting rule corresponds to a certain curvature, and in one of the shifted images, the features (of any kind) defining the lane boundaries are now arranged vertically. By vertically building the sum of the pixel values in each column of the shifted image, the columns containing the lane border features should show significant values (maxima or minima), which indicate the best matching curvature.

In order to find the best matching curve in the distribution map, similar to RALPH, the values of the distribution map are added column-wise with different lateral offsets for the different curve templates. In contrast to RALPH, the distribution map is not shifted, but a shift table is defined, which contains the offsets of each row for all tested curves. And according to this shift table, the distribution map values under a defined curve are added to build a curve rate. Figure 4.8 shows the curve templates.

Many feature classes provide a high probability for the vehicle trajectory on both sides of the feature, which means that their lateral offset histograms (compare section 2.2.3) show high values for both positive and negative offsets. In figure 4.4a, the effect of this double-sided offset distribution can already be seen, especially in the case of lane marking features. The resulting distribution map indicates high probabilities for several (in the best case parallel) curves. Sometimes there are actually multiple lanes visible leading to several possibilities for the best vehicle trajectory. This effect can be used to fit multiple models into the distribution map. For this purpose, not only curve templates starting at the vehicle position are tested, but also the same curve templates shifted to the left and

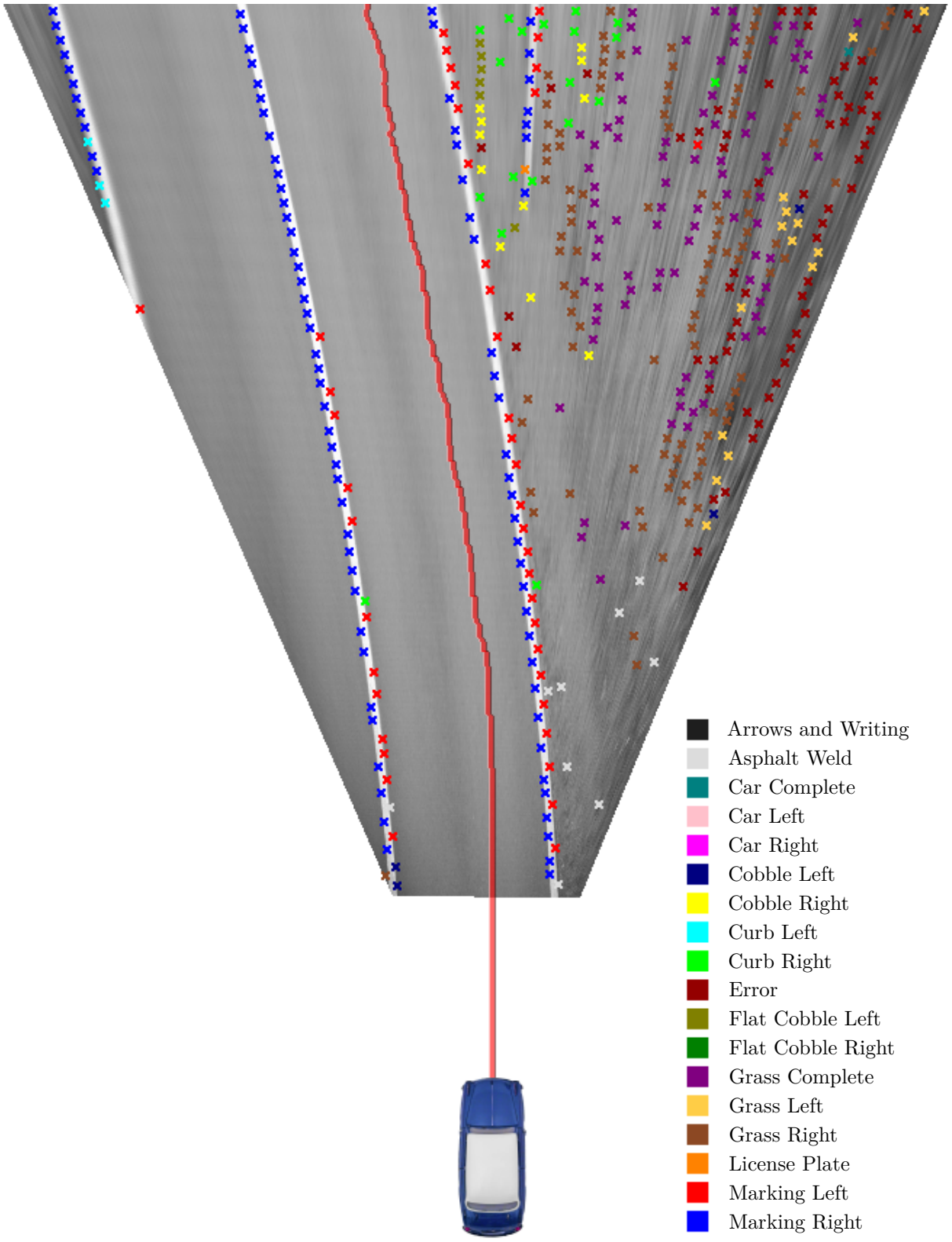


Figure 4.6.: Result of the dynamic programming approach without path smoothing

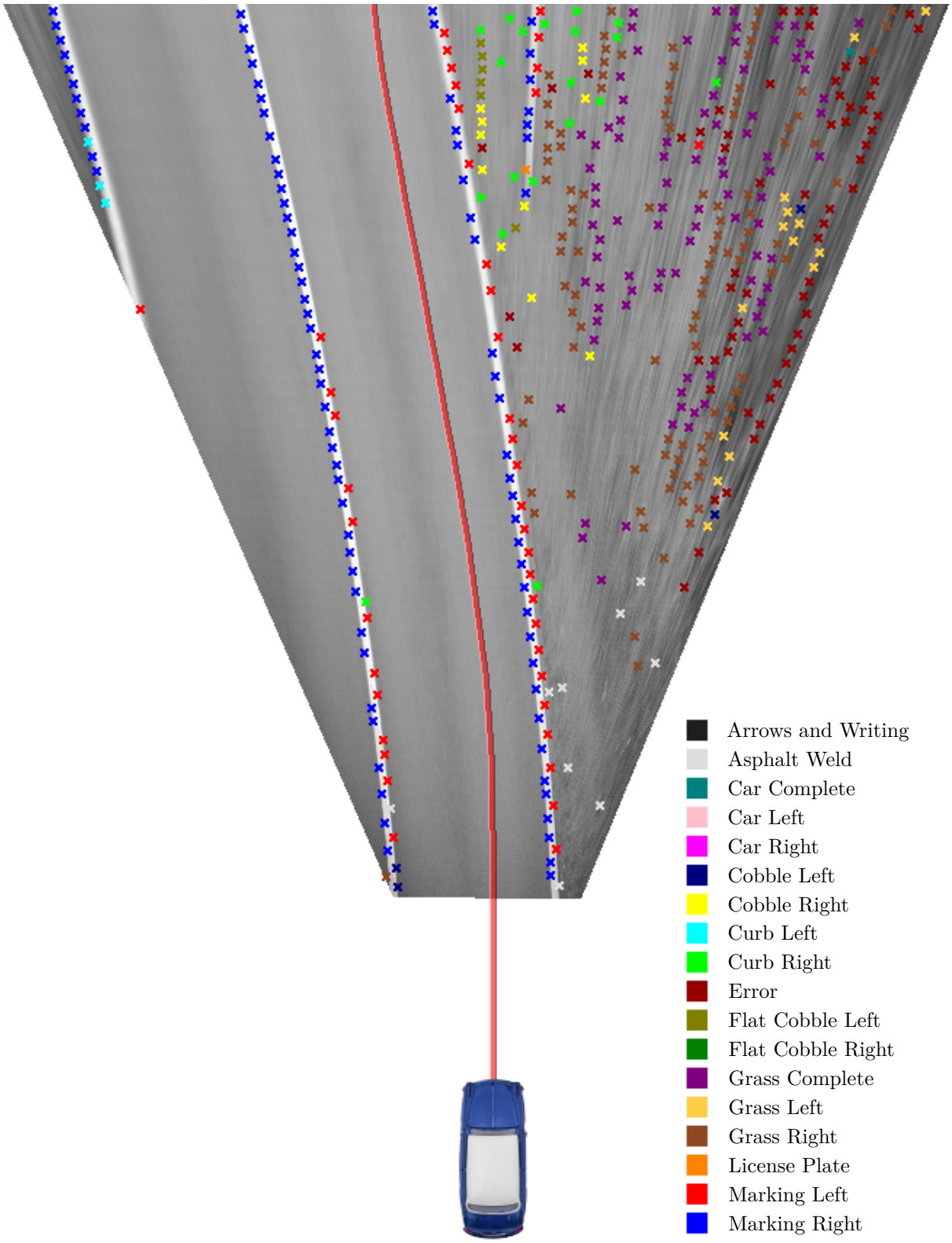


Figure 4.7.: Result of the dynamic programming approach with path smoothing

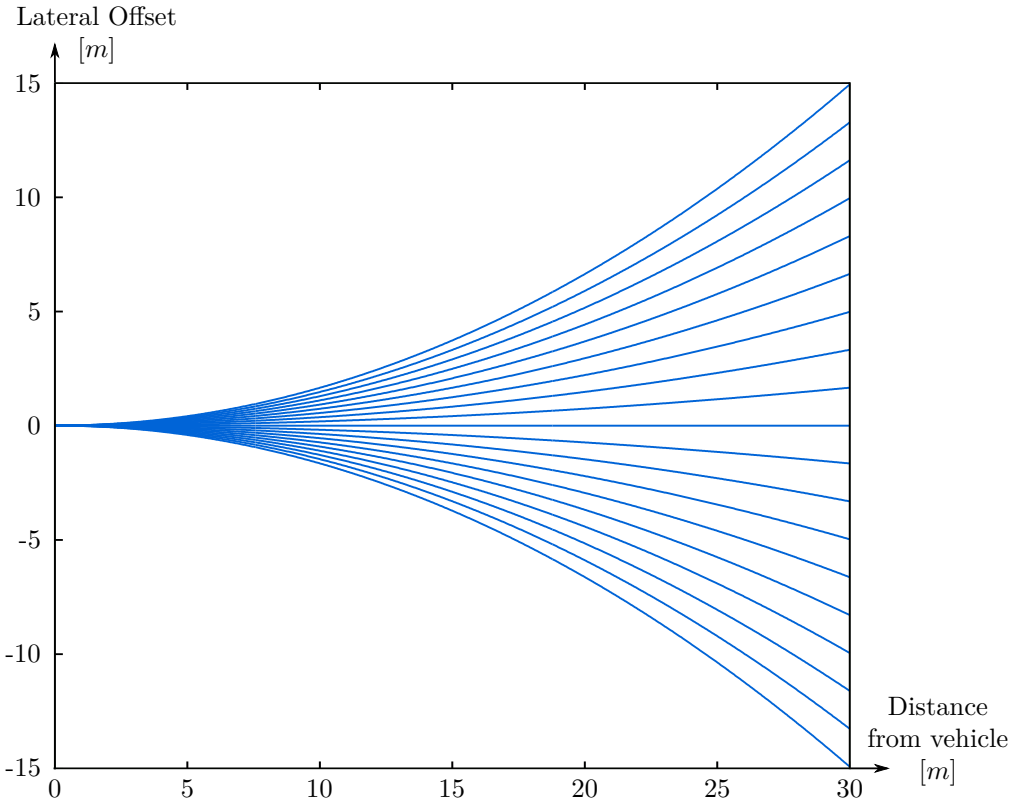


Figure 4.8.: Curve templates

to the right. This way, other lanes and distribution map maxima parallel to lanes are involved to increase the robustness of the model matching.
The curve templates are stored in a shift table, so that similar curvatures have similar indices. A curve can then be determined as a combination of several curves by calculating the average index. The resulting curve index $curve_{res}$ is determined by

$$curve_{res} = \frac{1}{sum(w)} \sum_{i=0}^{n-1} w_i \cdot bestCurveIndices[i] \,, \tag{4.4}$$

where n is the number of best curves used for calculating the result. The shift table indices of the n best rated curves are stored in the array $bestCurveIndices$, and each curve index is multiplied by a weight w_i, so the best rated template can, e.g., be assigned a higher weight than the second best, etc. The result is divided by the sum of all weights, $sum(w)$. With this method, the result can be a curve different from the best rated curve, of course, but the definition of the weights can limit the deviation from the best candidate. If, e.g., the indices are sorted by their rate in descending order ($bestCurveIndices[0]$ is the best rated curve index), the selection of weights $w_i = n-i$ ensures that the deviation from the best candidate stays small if not all other candidates prefer different trajectory curvatures. The resulting curvature is then used as a path starting at the vehicle position. Figure 4.9 and figure 4.10 show the results of the curve template matching with $n = 3$, which means that the three best rated curve templates are used to determine the resulting curve template.

In contrast to the previous matching approaches, the curve template matching works directly on the distribution map without the need of clustering or other intermediate processing steps.

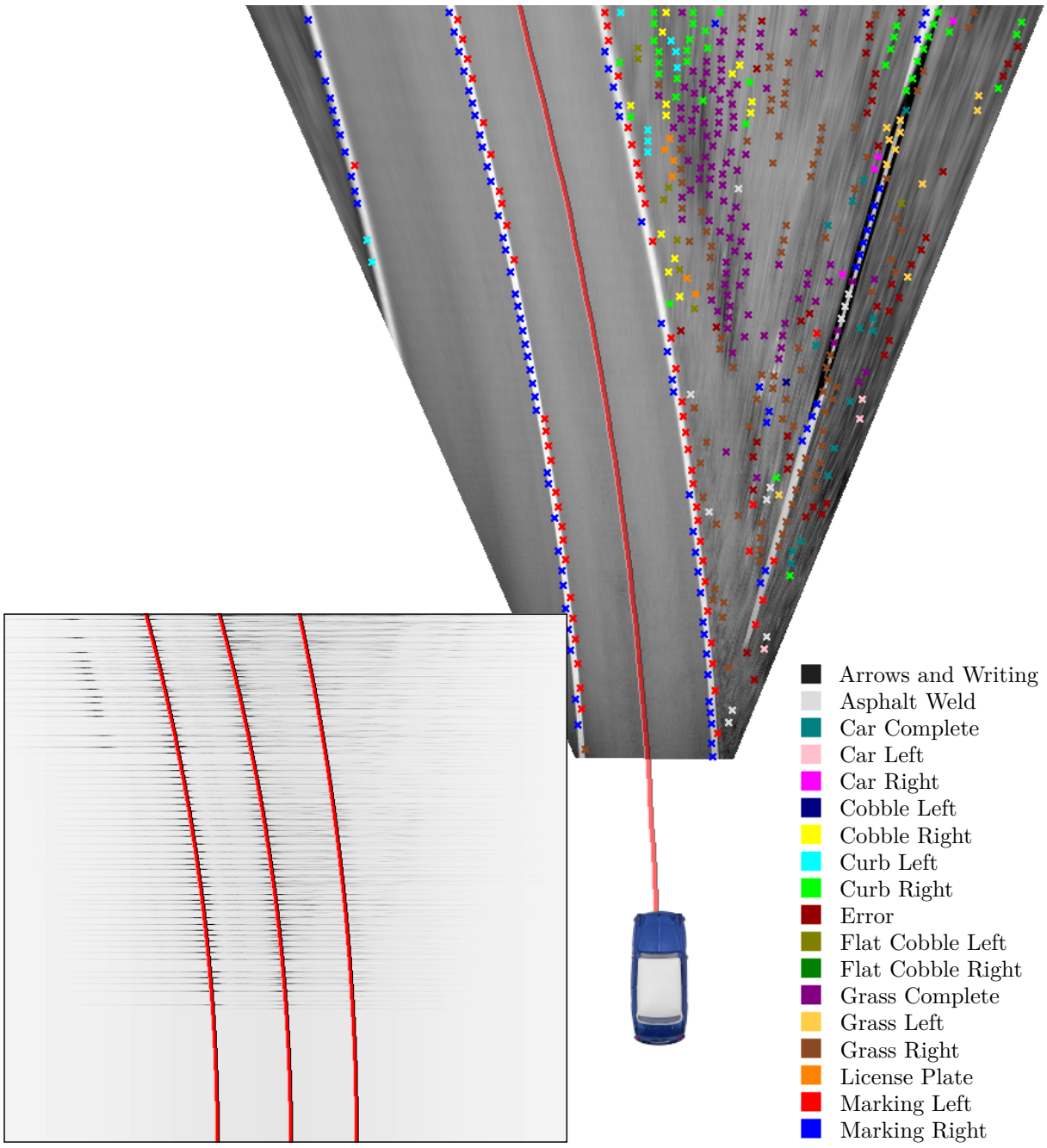


Figure 4.9.: Result of the curve template matching. The three best matching curves are combined to one resulting curve.

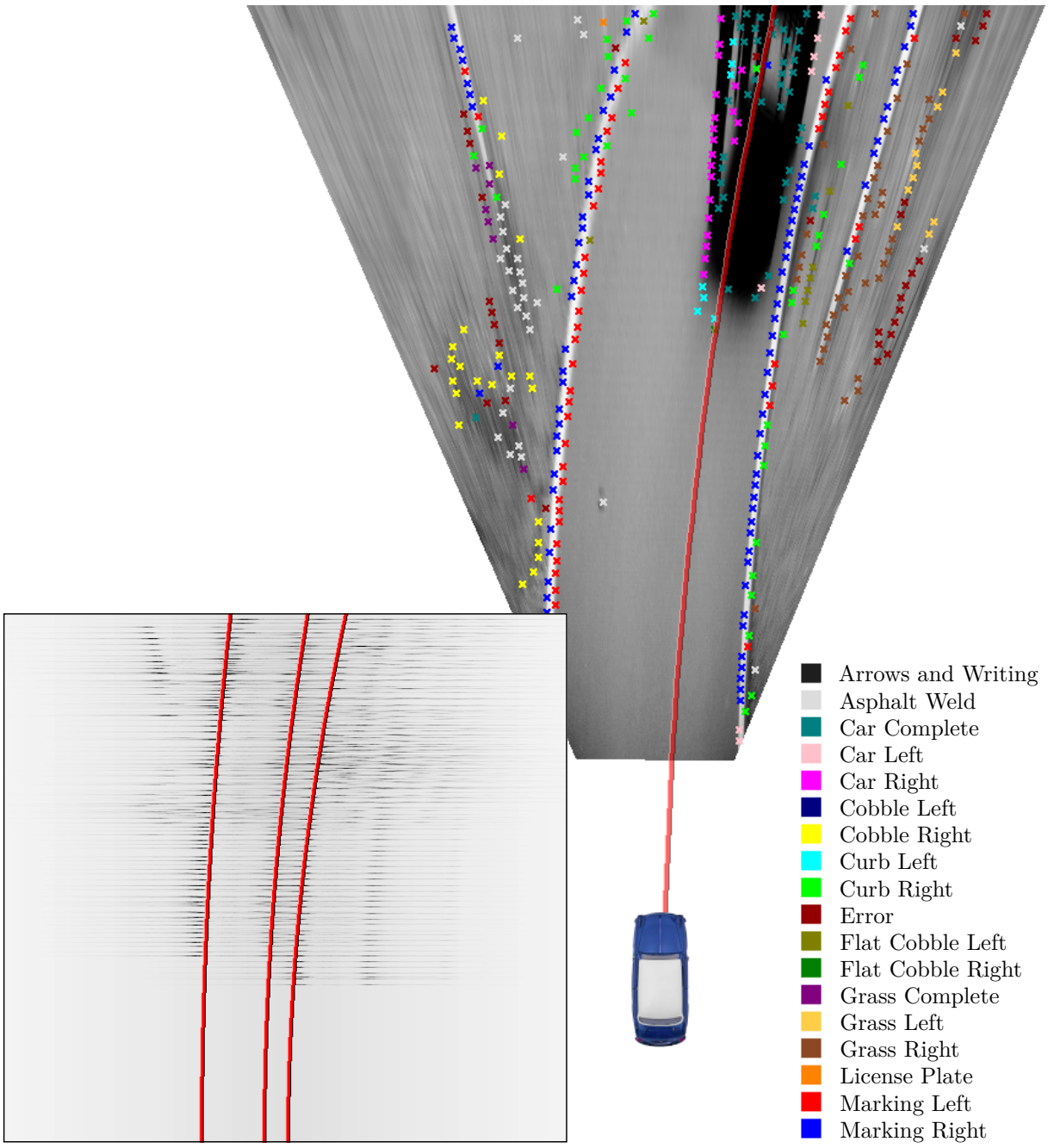


Figure 4.10.: Result of the curve template matching. The three best matching curves are combined to one resulting curve.

4.3. Model Selection

The previous sections presented three different approaches to the estimation of the vehicle trajectory. The question is which model should be used for the estimation of the trajectory, as depicted in figure 4.11. Although the result of the model matching already seems to represent a possible way of the vehicle, in many practical tasks it might be used as an initial solution to a driving corridor or lane border detection. This is why the geometric characteristics of the calculated path in terms of 'drivability' for a real vehicle are not an essential quality measure. More important are the computational efficiency, the robustness and the flexibility of the method.

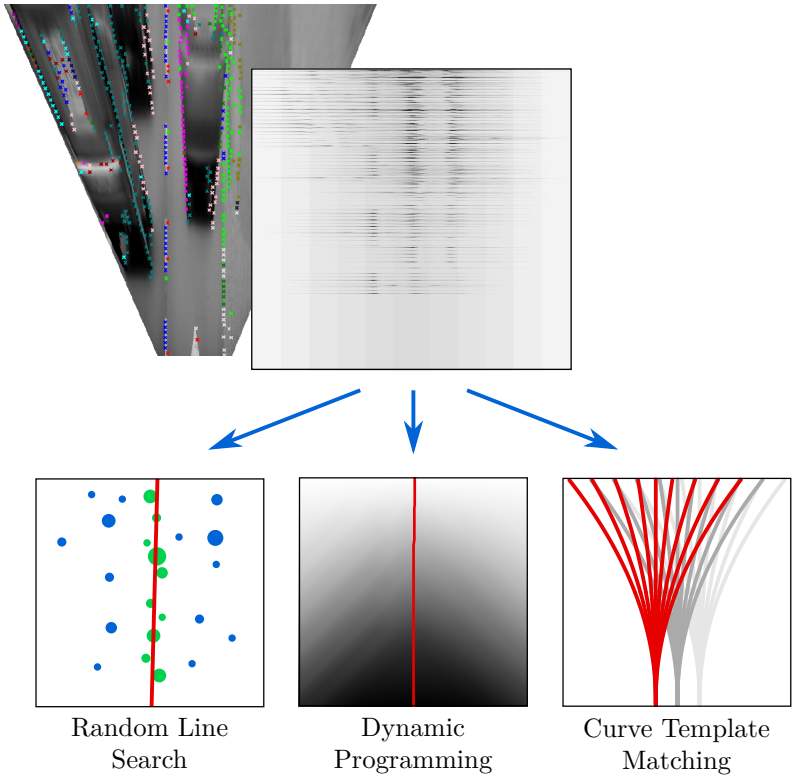


Figure 4.11.: Selection of the model matching method

The *random line search* (section 4.2.1) is fast, although the clustering needs some time, since the clusters are sorted by their weight. It is also robust against local disturbances in the distribution map, which means it provides realistic (coarse) estimates for the trajectory, even in cases of occlusions or false classifications. The robustness does not refer to the overall reliability of the model matcher. Since the result is a straight line, this model does not adapt well to complex lane courses, so it offers only poor flexibility. However, this does not mean that this model is not suited as an initial solution to further lane esti-

mation, even in winding road situations. Of course, RANSAC can also be used to detect curves, but here it is only applied to straight lines, since curves are already concerned by the other two matching methods.
The *dynamic programming approach* (section 4.2.2), on the other hand, is very flexible and can adapt to very complex lane courses. However, this characteristic decreases the robustness of this model matching approach, since the result can be distracted by local disturbances in the distribution map. Since this model is not geometrically restricted (except the maximum angle of 45° to the driving direction), its high flexibility can cause unrealistic trajectory estimations. A great advantage of the dynamic programming method is its efficiency, due to solving the originally recursive problem with an iterative algorithm.
In the case of the *curve template matching* (section 4.2.3), the computational efficiency and the flexibility depend on each other. Since the result is one of the predefined templates, a high number of templates is necessary to find the best curve for a given distribution map. And even with an extensive test of a high number of templates, the flexibility of this model is limited. On the other hand, the clear definition of possible solutions makes this model very robust against local disturbances in the distribution map, because the result of the model matching is always a realistic vehicle trajectory. Table 4.2 summarizes the characteristics of the presented model matching approaches. A good ('+') or very good ('++') quality regarding one of the properties *efficiency*, *robustness* and *flexibility* is highlighted green, and a bad quality ('−' or '−−') is highlighted red.

Table 4.2.: Characteristics of the model matching approaches

	Efficiency	Robustness	Flexibility
Random Line Search	+	+	−−
Dynamic Programming	++	−−	++
Curve Template Matching	−−	++	−

The three methods have different advantages and disadvantages. Figure 4.12 and figure 4.13 show some results of the different model matching methods. More example results can be found in appendix A (figures A.1 to A.3). These results already show that not all models are suitable for all possible situations. And there is also not one model, which is the best in all cases. In fact, the performance of the models depends on the situation. While sometimes a robust model seems to be preferable, in some situations a more flexible model can be necessary, and other situations might require a fast but still robust result. It finally depends on the context which model is the best. Thus, the selection of the appropriate model implies the determination of the context of the current scene. The definition and estimation of this *context* will be described in the next chapter.

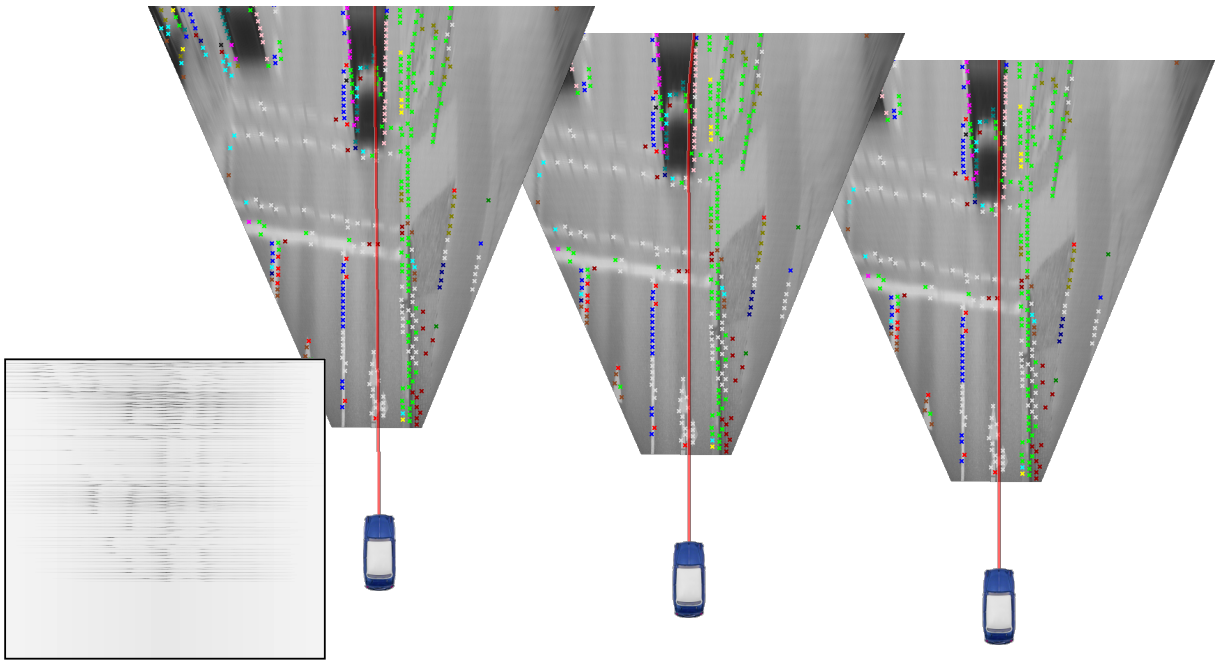


Figure 4.12.: Trajectory estimation with the distribution map and different models: random line search (left), dynamic programming (center), curve template matching (right).

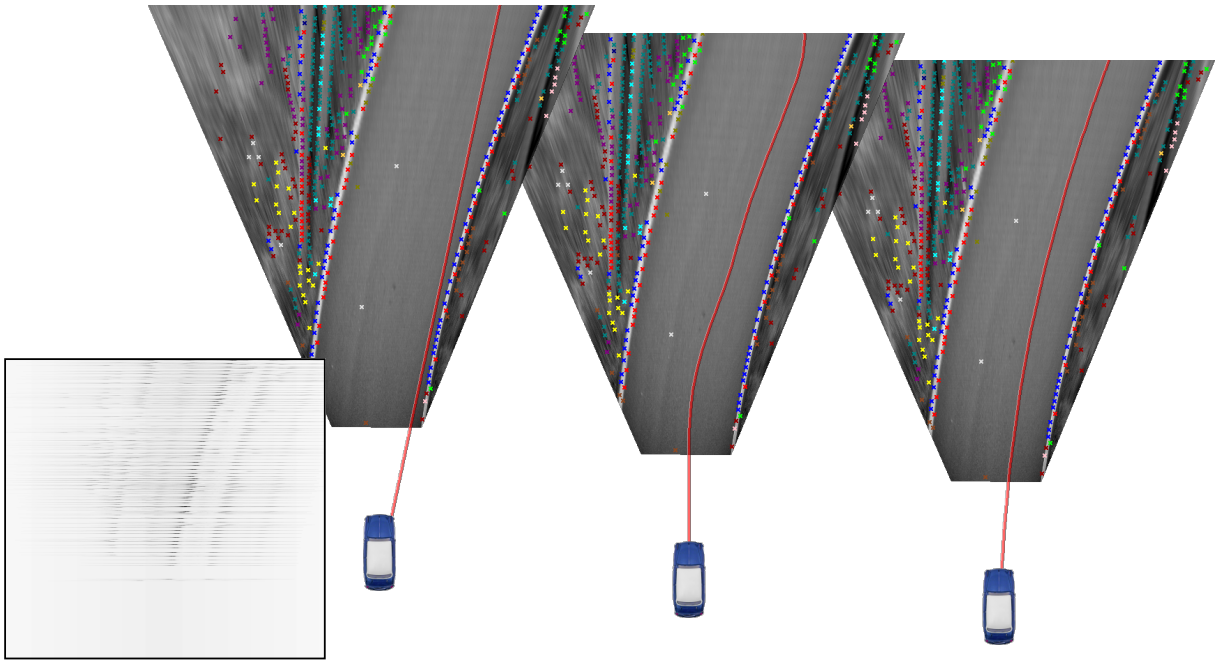


Figure 4.13.: Trajectory estimation with the distribution map and different models: random line search (left), dynamic programming (center), curve template matching (right).

Chapter 5

Using Context Information in Driver Assistance Systems

In the previous chapter, three different methods were presented to estimate the vehicle trajectory based on a given distribution map. It was suggested that the performance can be increased by an intelligent model selection, depending on the context. This chapter gives a short overview of the context definition in image processing, shows how the context can be defined in traffic scenarios, and how to determine the context of the current situation.

5.1. Context Definition in Image Processing

For the task of object detection, Hoiem et al. use the context to consider the fact that detection certainties of objects influence each other (Hoiem et al., 2008). The successful detection of a road, e.g., increases the detection probability of a car and vice versa. They define the context as the detected objects and a rough approximation of the 3D structure of the scene, which allows a location dependent modeling of the object appearance probabilities. The advantage is, that for an object of known dimensions, its size in the camera image can be predicted, which simplifies the detection.
Oliva and Torralba explore the shape of a scene by determining several scene characteristics, like *openness*, *perspective*, *symmetry*, etc (Oliva and Torralba, 2001). To calculate these characteristics, they apply a *Fourier Transform* to analyze the spectral signature of images of different scene categories.
Another approach to image categorization was presented by Lazebnik et al. by building a resolution pyramid, dividing the image into subregions and creating histograms of local features for each region (Lazebnik et al., 2006). The feature vectors, generated by concatenating the single histograms of each region of the different pyramid levels, have a size up

to 34000 for 4 division levels (1×1 to 8×8) and a vocabulary size of 400 (the local features are clustered to 400 possible values). The resulting feature vectors are then classified by a support vector machine.
Faktor and Irani have developed an unsupervised image categorization by clustering images (Faktor and Irani, 2014). Within a cluster, each image can be composed by components of other images of the same cluster.
In the area of lane detection, Shang et al. use contextual information in form of the position in a given map, provided by GPS (Shang et al., 2013). They use the map information to predict the course of the road and build a confidence map which is combined with the result of a texture analysis to improve the segmentation of unstructured roads in camera images. Alvarez et al. combine different contextual information, e.g., scene layout, vanishing point and horizon line, within a Bayesian framework for road segmentation (Alvarez et al., 2010).

Although the definitions of context differ strongly, they all have in common that the contextual information has a global character, in contrast to local features which describe only a certain region in the image. Several types of information can be summarized as context, e.g., the overall distribution of local features in an image (Hoiem et al., 2008; Lazebnik et al., 2006), or image characteristics describing the complete scene (Oliva and Torralba, 2001), which also includes the location of a certain entity within the image (e.g., vanishing point) (Alvarez et al., 2010). This contextual information is described by global image features. Furthermore, *external* information can be part of the context. This includes, e.g., the position of a vehicle inside a map (Shang et al., 2013), information about weather and time, or other available data.
The usage of context in image processing ranges from the support of object detection to supervised or unsupervised scene categorization. In this work, contextual information is used to support the camera-based lane detection. How this contextual information is determined, will be described in the next section.

5.2. Context Classification

This section describes which contextual information is used to support the lane detection. After defining context classes, global features are constructed to determine the context of the current traffic scene.

5.2.1. Context Classes

Section 5.1 already gave some examples of how the context can be defined in image processing tasks. Although unsupervised learning approaches allow the generation of abstract context classes (e.g., image categorization by clustering), it can be helpful to define concrete (semantic) context classes with a certain semantic meaning (compare section 3.1 for the discrimination between abstract and concrete features). Contextual information with a semantic meaning allows an easier further use of this information in other scene-dependent algorithms. Besides the camera-based lane detection, other driver assistance systems might be configured depending on the situation, that means on the current traffic scenario. To provide this information, 4 typical traffic situations are chosen: highway, crossroads, country roads, and urban scenario. These are the context classes which can be assigned to every camera image. Table 5.1 shows some example images of the 4 different context classes.

Table 5.1.: Context classes with example images

Highway

Crossroads

Country

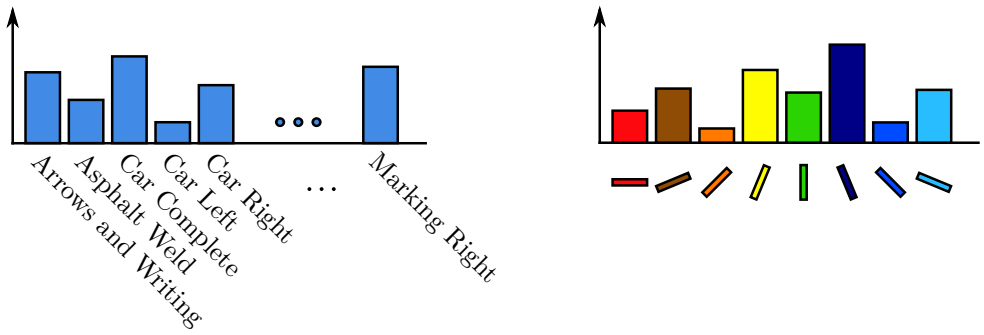
Table 5.1.: (continued)

Urban

5.2.2. Global Features

To determine the context class of the current scene, according to section 3.2, features are needed as the input data for a classification algorithm. And in contrast to the local features which represent certain locations in the image (see chapter 3), the calculation of the global scene information, the context, requires global features. Just like the local features (and also the context classes, as described in the previous section), these global features can also be grouped into abstract features and semantic features. Abstract features describe characteristics of the whole image (or parts of it) which do not provide apparent information to the observer (or only partially). Examples for abstract global features are primarily histograms of certain properties, e.g., of gray values, colors or gradient directions. Semantic features, on the other hand, do provide concrete information. These include external information, e.g., the current time, weather, or a location within a map, which can help estimating shadows, glare, or upcoming road curvatures. Of course, some of these features could also be considered already as a context class. The distribution of the local feature classes in the current scene, represented by a histogram, is a global feature which can be arranged between abstract and semantic features. While the histogram actually has an abstract character, some entries might already provide direct information about the context class, but this depends on the defined classes of both the context and the local features. A high number of cobblestone pavement features (e.g., *Cobble Right*) excludes, e.g., a highway scenario.

For the classification of the context, a combination of two global feature types was chosen to build the feature descriptor. The first part is built by the histogram of local features (figure 5.1a; see chapter 3 for the complete set of local feature classes). This histogram is normalized, so it does not depend on the total number of detected features, which makes the descriptor less affected by changes in the contrast characteristics of the camera and also invariant against the size of the observed area (bird's eye view image).
The second part is a Histogram of Oriented Gradients (Dalal and Triggs, 2005), or rather 4 histograms. To estimate the context class, the original camera image has a more informative value than the BEV image, because only a small portion of the camera image is



(a) Histogram of local features **(b)** Gradient orientations

Figure 5.1.: Descriptors of the global features for context classification

projected to the ground. The image contents above the ground plane are too important to be disregarded. For this reason, the camera image is divided into 4 subregions, and after a Canny Edge detection (see section 3.1.1), for each of these quadrants, a histogram of orientated gradients (with 8 possible orientations) of the detected edges is generated and, of course, normalized, as well. Figure 5.1b shows an example histogram. While the lower regions usually contain structures like curbstones, lane markings, grass, etc., the upper quadrants exhibit sky, trees, or buildings. Figure 5.2 shows the different gradient directions in the 4 subregions of the image, marked by different colors, according to the colors in figure 5.1b.

The 18 values from the local feature histogram and the 8 values from each of the 4 gradient orientation histograms are concatenated to a feature vector with 50 elements which will be used as the input vector for a classifier in the next section.

5.2.3. Classification

The training of classifiers and the classification of samples with given feature descriptors was already discussed in chapter 3. The same way the local features are classified, the context can be determined using the feature descriptor described in the previous section. To evaluate the context classification, 783 samples from the complete available training sequences were manually classified into the 4 context classes *Highway* (162 samples), *Crossroads* (173), *Country* (141), *Urban* (307). Again, the performance of a selection of classifiers was tested. Table 5.2 shows the portion of correctly classified samples in a 10-fold cross-validation (see chapter 3). The two parts of the feature vector, the local feature histogram (LFH) and the histogram of oriented gradients (HOG), were tested separately and in conjunction as the final feature vector (LFH+HOG).

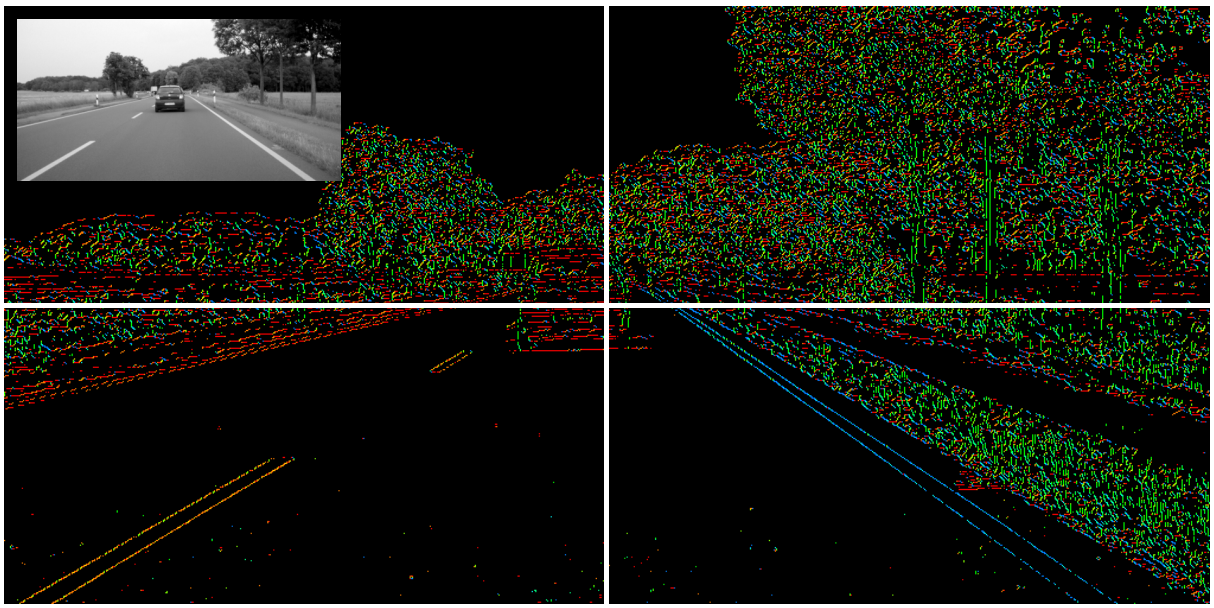


Figure 5.2.: Edges marked with different colors, depending on their gradient orientation, according to the colors in figure 5.1b. The original camera image is displayed in the upper left corner.

Table 5.2.: Classification performance in % of correctly classified samples (10-fold cross-validation)

Classifier	LFH	HOG	LFH+HOG
Nearest Neighbor	93.0	98.7	99.0
Logistic Model Trees	88.9	96.7	97.6
SVM (linear kernel)	82.9	88.1	94.8

The results in table 5.2 seem very promising, but their significance should not be overrated. Although the number of samples of each class is passably balanced, this number does not correspond to the number of really different scenarios, especially regarding the class *Crossroads*. Because the 173 crossroad samples were extracted from only 5 different crossroad scenes, from different locations at different time steps. This affects the result of the cross-validation, since there are typically some samples in the remaining training portion, very similiar to the samples in the test portion (from the same scene). However, this problem primarily applies to the crossroads class, since the training set contains far more different highway, country, and urban scenarios. Anyway, experiments with other test samples showed a good performance of the context classification (figures 5.6 and 5.7). Figures 5.3 to 5.5 show some example results of the context classification. As in chapter 3, Logistic Model Trees were used for the classification, since they provide a good classification performance as well as the class probabilities for all context classes, and not only the best matching class. Apart from the probabilities for the context classes, the figures contain the camera image, the bird's eye view image with classified local features, and the resulting feature vector, composed of the local feature histogram and the histograms of oriented gradients for the 4 image quadrants. More example results can be found in appendix B (figures B.1 to B.5).

How the classified context can be used to improve the performance of the lane detection, will be described in the next chapter.

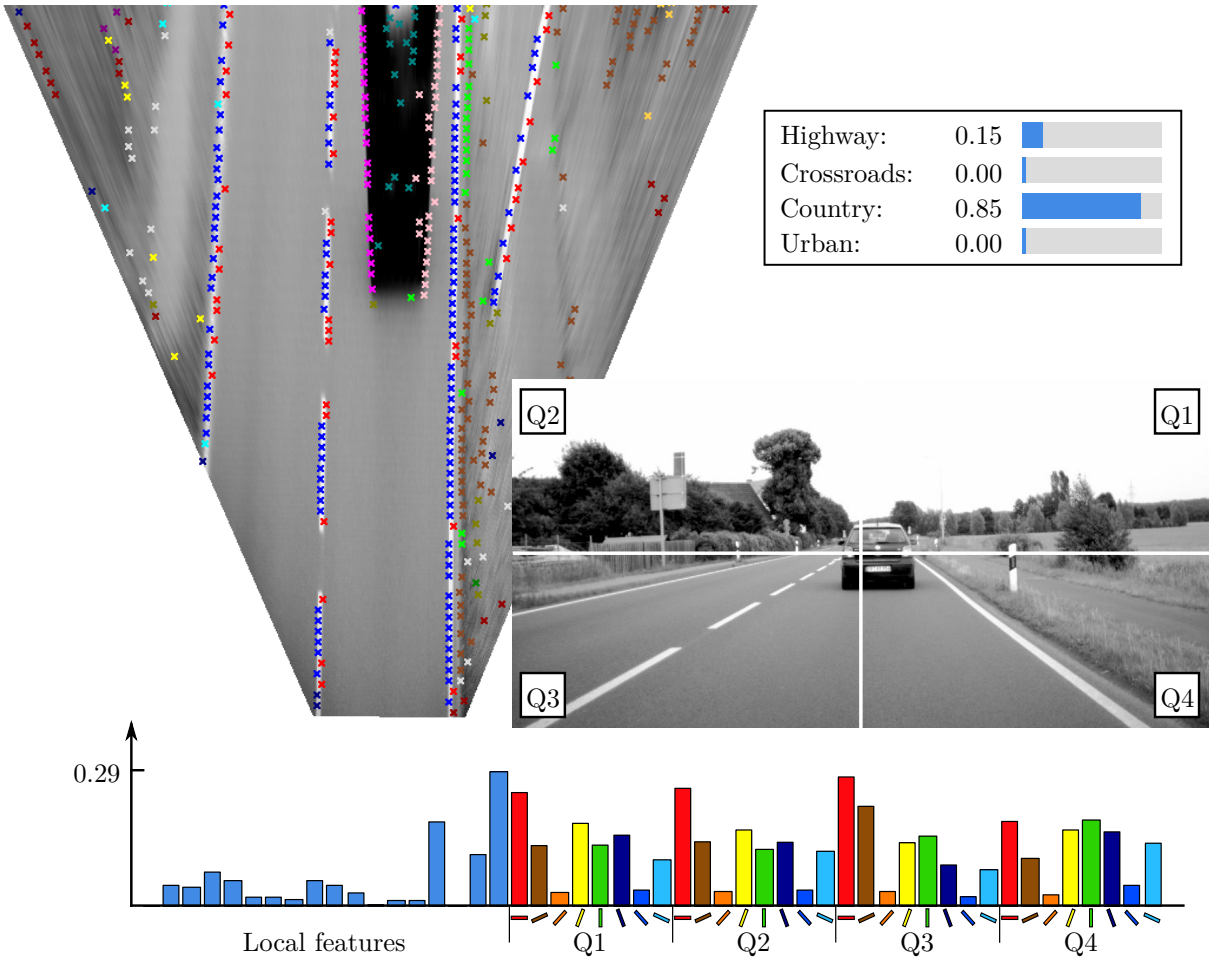


Figure 5.3.: Context classification result: camera image with subregions Q1-Q4, BEV image with local features, and the global feature vector

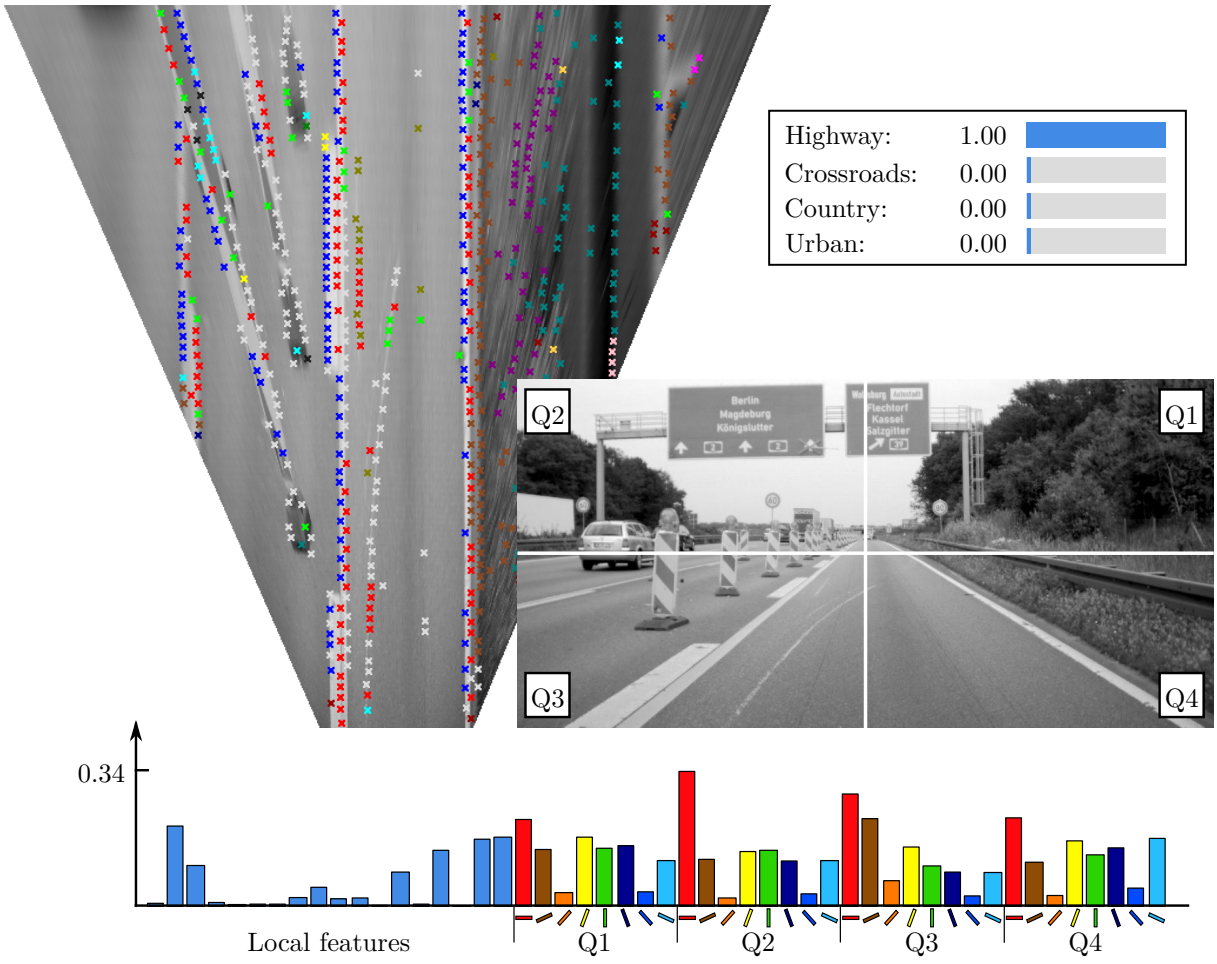


Figure 5.4.: Context classification result: camera image with subregions Q1-Q4, BEV image with local features, and the global feature vector

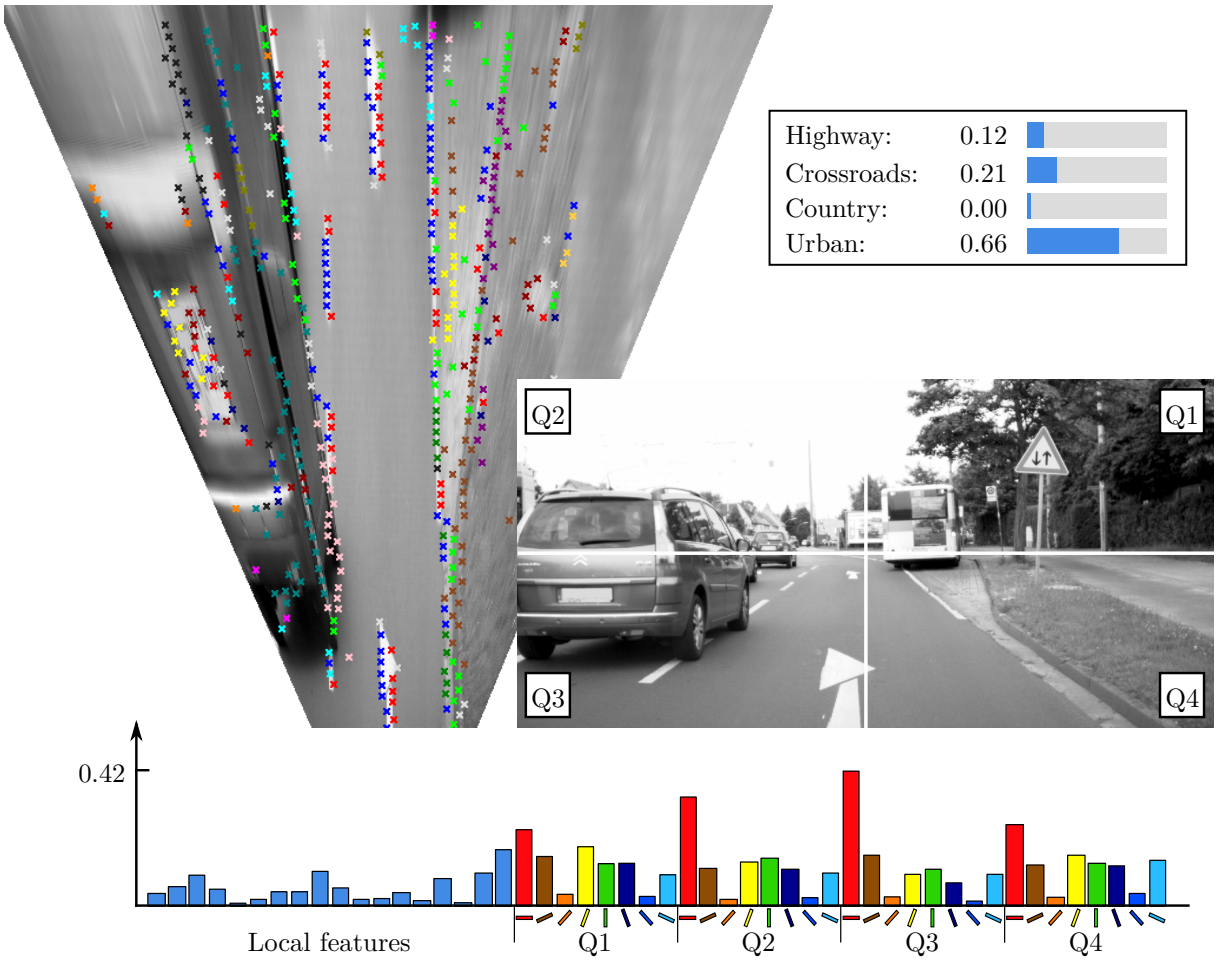


Figure 5.5.: Context classification result: camera image with subregions Q1-Q4, BEV image with local features, and the global feature vector

Figure 5.6.: Context classification result with image from other dataset

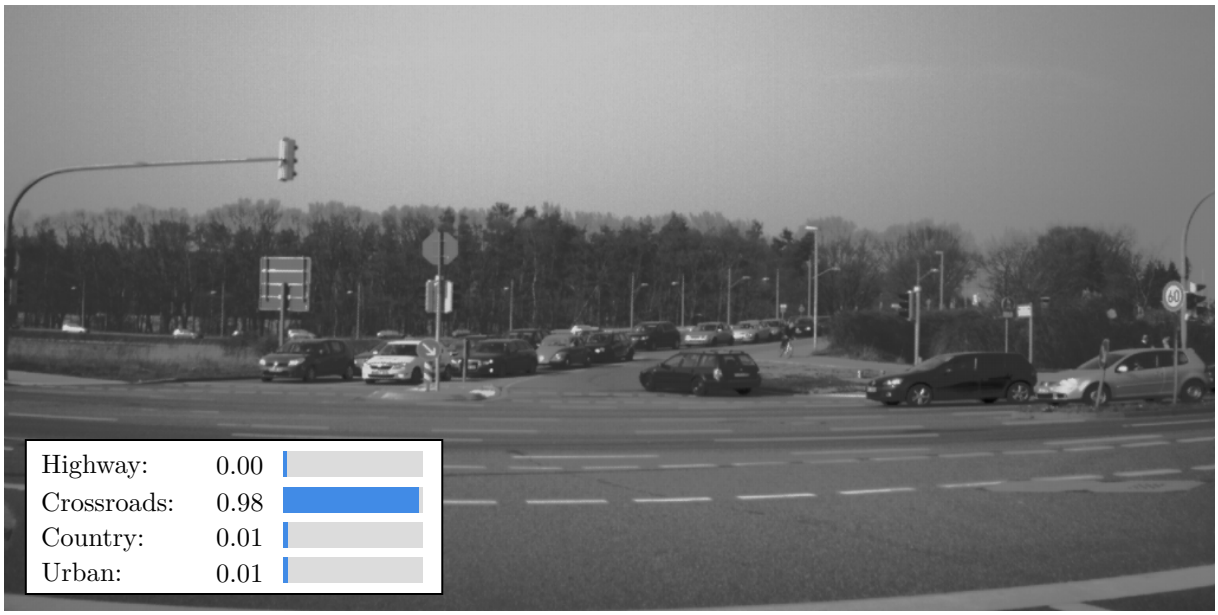


Figure 5.7.: Context classification result with image from other dataset

Chapter 6

Applications and Evaluation

The previous chapters described how local image features can be classified and used to predict the vehicle trajectory, and how global features can be used to determine the context of the scene. Since this framework could be used as a part of a whole System for automated driving, the obtained features and context classes could be used for parameterizing other system components as well.
In this chapter, some application examples of the context classification for lane detection are described and the results are discussed. The context determination allows a situation dependent parameter adaptation of driver assistance systems, or even a selection of active algorithms for automatic driving in different situations. Of course, many more context classes could be considered which is beyond the scope of this work. However, future work could also regard classes like night, rain, highway exit, traffic jam, etc. After explaining the dataset used for training and evaluation of the proposed method, the following sections describe how the knowledge of the context can improve the performance of the trajectory prediction. Based on the context, the best model can be selected, the spatial relations can also be learned for different context classes, and even the local features can have a different importance in different scenarios. A direct performance comparison to other methods is not possible since most approaches detect the lane by image segmentation (which pixels belong to the road?) or lane marking detection (where are the lane boundaries?). In contrast to these approaches, the proposed framework estimates the course of the lane, regardless of the type of its boundary. But the last section of this chapter discusses how the presented trajectory prediction can support a real lane or lane border detector.

6.1. Dataset

The dataset used for training and testing this approach contains 22 sequences of different traffic situations. These sequences were separated randomly into two subsets, one for learning the spatial relations (training), and one for testing. These subsets consist of 11 disjoint sequences, so each sequence is only part of one subset. The training set contains 10.674 images and the testing set consists of 12.395 images.
Not only for learning the spatial relations, but also for the evaluation, the driven trajectory is used as ground truth. Hence, it has to be mentioned that the dataset contains lane changes which cannot be considered correctly for the evaluation, because during the lane change maneuver, the vehicle is driving between two lanes, while the proposed method still estimates approximately the center of the lane. While lane changes in the training set do not cause problems if there are enough normal driving situations, every lane change in the testing set decreases the quality of the result, because the ground truth (the real trajectory of the vehicle) does not represent a real lane for multiple consecutive frames. While figure 6.1 shows some typical images from the dataset, figure 6.2 illustrates some problems of using complete sequences, without excluding critical situations. Apart from the lane change, where the vehicle is driving between two lanes, the narrow viewing angle of the camera also leads to difficulties in certain situations. Especially in urban scenarios, the main part of the road can be occluded by vehicles driving ahead. Furthermore, in case of turns or narrow curves, the camera cannot see the ground truth lane, so it is very unlikely that this lane is detected correctly. Especially in scenarios with ambiguous driver intentions, e.g. at crossroads, a vehicle's turn cannot (and should not) be predicted if only the lane in forward direction is visible in the image and no further information is available. This issue of using the raw or a preprocessed dataset is also discussed in chapter 7.1.

Figure 6.1.: Examples from the dataset

Figure 6.2.: Examples from the dataset: Lane changes, occluded road, right turns into areas not captured by the camera (bottom row)

6.2. Context-based Lane Model Matching

In this section, the selection of the lane model matching method based on the context class, according to figure 4.11, is analyzed. The basis for this experiment is the assumption that not one matching method is the best in all situations, but that in one context a certain lane model might be better, while in another context a different model leads to better results. If for each context class the most suitable lane model matching method can be determined in the training set, the overall error should be decreased if the same model selection strategy is used in the testing set. Section 4.3 already discussed the three matching methods *Random Line Search*, *Dynamic Programming*, and *Curve Template Matching*, as well as their advantages and disadvantages. To determine the best matching method for a context class, the errors of the different matching methods for all context classes were calculated. Before analyzing the results, it has to be discussed what is the error between the estimated and the ground truth trajectory.

There are many ways to describe the difference between two arbitrarily shaped curves. The ground truth trajectories consist of single points (positions at certain time steps, synchronized to the frames of the image sequence). The estimated trajectories can also be represented by single points, so the difference between two polylines has to be calculated. One possibility is to calculate the maximum distance between the two curves. For this purpose, for each point p of the estimated curve c, the minimum Euclidean distance between its coordinates $X_{c,p}$ and the corresponding ground truth trajectory $\tilde{\tau_c}$ is calculated. The squared maximum of all these smallest distances is considered as the squared error between the estimated and the ground truth trajectory. Analyzing the complete training set consisting of m samples (curves) leads to a set of squared maximum deviations. The median *medSE* of these squared errors is defined by

$$medSE = \underset{c=1..m}{\text{median}} \left(\left(\max_{p=1..n} \Big(\text{minDist}(X_{c,p}, \tilde{\tau_c}) \Big) \right)^2 \right) , \tag{6.1}$$

and the errors for the different context classes and model matching methods are listed in table 6.1. The dynamic programming approach, e.g., applied to the training samples classified as *country*, results in a median squared error of 0.19, while the line search approach for the complete training set leads to an error of 0.66 (last row).
Another way of describing the difference between the two curves is to calculate the area between them. However, instead of calculating the exact area between the curves, the sum of the minimum distances between the points of the estimated and the ground truth trajectory are an adequate representation of this measure. So, the smallest absolute distances of each point of the estimated trajectory to the ground truth trajectory are added and normalized (divided by the number of points). The mean of these errors for the complete

training set is referred to as the mean normalized integrated error *MNIE* and is listed in table 6.2. This error is a good measure for the area between the two curves:

$$MNIE = \frac{1}{m}\sum_{c=1}^{m}\left(\frac{1}{n}\sum_{p=1}^{n}\Big(\text{minDist}(X_{c,p},\tilde{\tau}_c)\Big)\right). \tag{6.2}$$

Besides the integrated or maximum distances between the curves, for some applications it might be more important how many of the m solutions show an error (maximum distance to ground truth curve) below a certain threshold ϵ and can therefore be considered as a correct result. The portion of correct solutions *PCS* is defined by

$$PCS = \frac{\#\left\{c \in C \mid \max_{p=1..n}\Big(\text{minDist}(X_{c,p},\tilde{\tau}_c)\Big) < \epsilon\right\}}{m}, \tag{6.3}$$

where C is the set of all samples (curves) in the training set with $\#C = m$ the number of samples.
However, the selection of this threshold is difficult and might again depend on the application itself. Considering the lane width of different road types in Germany, the evaluation was proceeded with two different thresholds, one set to $\epsilon = 1.4$ meters and the other one to $\epsilon = 0.7$ meters, which is approximately a half, respectively a quarter of the minimum lane width in Germany (the lane width lies between 2.75 and 3.75 meters). The portion of correct solutions, with a maximum distance between the curves below these thresholds is listed in table 6.3 and 6.4. It has to be mentioned that for the number of correct solutions, often the term *true positives* is used. However, the calculation of true positives makes more sense, e.g., in pixel classification tasks, where beside the true positives, classified pixels can also be false positives, as well as true and false negatives. If a complete image segmentation, e.g., an assignment of each pixel to road or no road, is performed, the number of true and false positives and negatives can be used to calculate other error metrics, e.g., the f-measure (Fritsch et al., 2013). For describing only the correctness of a solution, like in this case, the term *portion of correct solutions* is more suitable.
This section only shows a small selection of possible error metrics. It depends on the application which metric should be applied. Tables 6.1 to 6.4 show the errors for the training data. Choosing one error metric (or evaluation measure) suitable for a certain application, allows the selection of the optimum model matching method for each context class, with the smallest error or the highest number of correct solutions.

The bold green values in tables 6.1 to 6.4 represent the best values (smallest errors) for each context class. These can be used to select the best model matching methods for a given context, depending on the desired error metric. Choosing, e.g., the mean normalized

integrated error (table 6.2), the evaluation with the training dataset suggests using the *line search* in crossroads and urban scenarios and the curve template matching in highway and country situations. Applying this model matching scheme to the testing set decreases the overall error (*MNIE*) by 10% compared to the smallest error of one model matching for the complete testing set.

Figure 6.3 shows the accuracy of the results depending on the distance from the vehicle (up to 30 meters) for the context-based lane model selection according to table 6.2 (mean normalized integrated error). The plot shows the mean deviation between the estimated and the ground truth curves for the complete testing set, where the deviation is the minimum Euclidean distance between a point of the estimated polyline at a given distance from the vehicle (horizontal axis) and the ground truth trajectory. The mean deviation, e.g., at a distance of 20 meters from the vehicle is approximately 0.5 meters.
This section showed how the context-based lane model matching can improve the performance of the lane detection. Depending on the application, the appropriate evaluation measure (error metric) has to be chosen. The following sections discuss further possibilities for using the contextual information in the lane detection framework.

Table 6.1.: Median squared error in the training set

	Line Search	Dynamic Programming	Curve Templates
Highway	0.80	1.36	**0.35**
Crossroads	8.61	8.01	**4.61**
Country	0.66	**0.19**	0.24
Urban	0.50	**0.38**	0.42
All	0.66	0.42	0.34

Table 6.2.: Mean normalized integrated error in the training set

	Line Search	Dynamic Programming	Curve Templates
Highway	0.22	0.38	**0.18**
Crossroads	**0.58**	1.80	1.82
Country	0.13	0.17	**0.10**
Urban	**0.15**	0.22	0.20
All	0.21	0.41	0.30

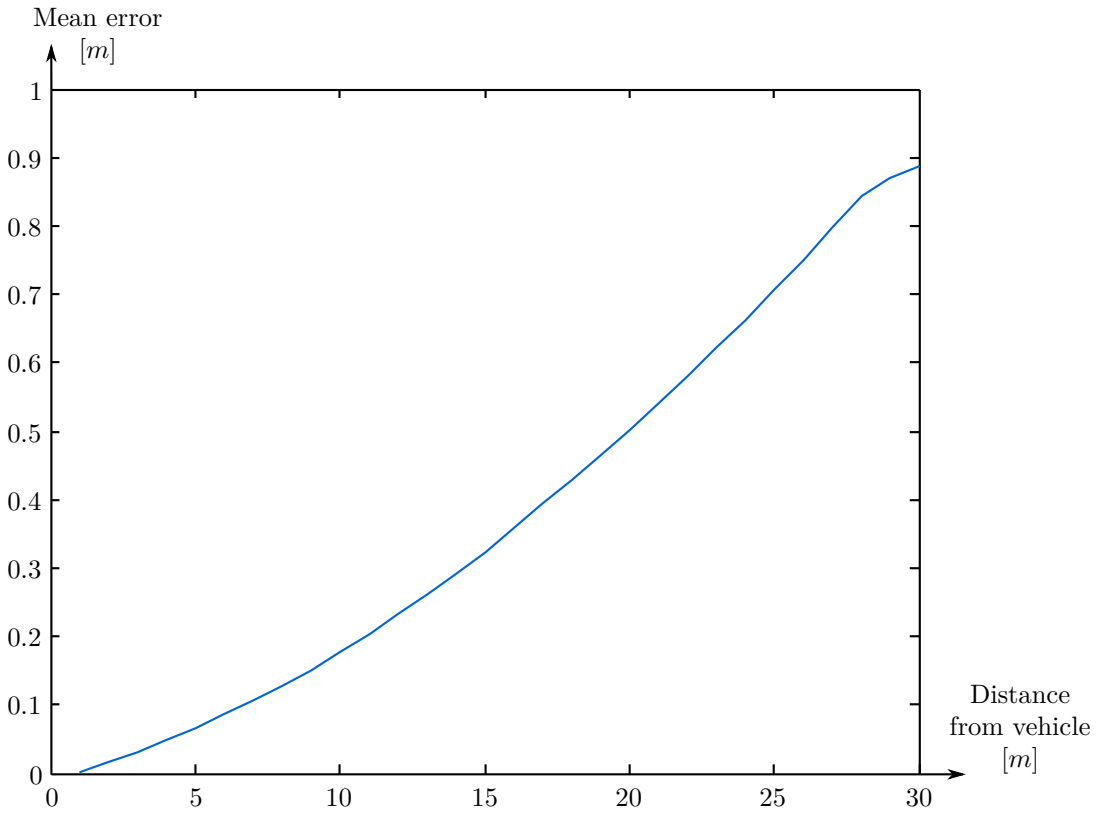


Figure 6.3.: Mean deviation between estimated and ground truth trajectory

Table 6.3.: Portion of correct solutions in the training set (threshold: 1.4 meters)

	Line Search	Dynamic Programming	Curve Templates
Highway	0.72	0.51	**0.85**
Crossroads	**0.44**	0.36	0.34
Country	0.92	0.90	**0.96**
Urban	**0.92**	0.88	0.90
All	0.82	0.74	0.85

Table 6.4.: Portion of correct solutions in the training set (threshold: 0.7 meters)

	Line Search	Dynamic Programming	Curve Templates
Highway	0.43	0.33	**0.64**
Crossroads	**0.34**	0.26	0.33
Country	0.57	0.81	**0.86**
Urban	0.65	**0.70**	0.68
All	0.55	0.58	0.68

6.3. Context-based Spatial Relations

Not only the lane model can be selected for a given context class. In chapter 2, the learning of the spatial relations between different features and the vehicle trajectory was described. These spatial relations are learned with the complete training set. However, it can be assumed that the spatial relations might differ from one situation to the other. E.g., grass might have a different location relative to the vehicle trajectory on a country road than in an urban scenario. For this reason, this section analyzes the application of context-based spatial relations.
For each context class, the spatial relations are learned independently for all local feature classes. So, instead of 18 lateral offset histograms (see section 2.2.3) for the 18 local feature classes, 72 histograms are obtained, because for each feature class, e.g., *Arrows and Writing*, one histogram for each of the 4 context classes *Highway*, *Crossroads*, *Country* and *Urban* is generated. To build the distribution map in the testing set, the spatial relations for each local feature corresponding to its feature class and to the classified context are applied. However, the use of the context-based spatial relations only results in a slight improvement, e.g., the portion of correct solutions for different thresholds is increased by 1%, which is not statistically significant. An analysis of the lateral offset histograms of the local features for the different context classes showed that in general, the spatial relations are very similar for all context classes. One exception is the context

class *Country*, which contained a greater difference of the lateral offset histograms from the other context classes. The reason might be that country roads have a more constant number of lanes, usually one lane per direction, while in all other scenarios the number of lanes varies between 1 and 5 lanes per direction.

6.4. Context-based Feature Weighting

In the previous sections, the different local feature classes were used as a fixed given set, and their lateral offset histograms were used to build the distribution map (see section 2.3.1). Until this point, it has not been analyzed which feature classes are more important for the lane estimation and which ones can be omitted without a loss of performance or maybe even resulting in an improved distribution map quality. The appearance of the histograms (see figure 2.8 on page 18) and the distribution map (figure 2.9b on page 21) show that a histogram with significant peaks supports the lane model matching better than a uniformly distributed histogram. For this reason, the significance ζ of a lateral offset histogram h can be measured by the normalized sum of squared values:

$$\zeta(h) = \frac{1}{n} \sum_{i=1}^{n} h_i^2 \,, \tag{6.4}$$

where n is the number of entries in the histogram and h_i is the i-th element of the histogram. This measure leads to a low value for uniform distributions and to a higher value if significant peaks are contained in the histogram. Figure 6.4 shows the significance values for the spatial relations of the local feature classes for the different context classes.

There are feature classes with a low significance in all context classes (e.g., *Car Complete*), and there are local features with differing significances, depending on the context. The feature *License Plate*, e.g., seems to provide useful information only in crossroad scenarios. A possible explanation is that in many crossroad situations, another vehicle is standing right in front of the camera, so its license plate is detected reliably at many time steps in the center of the lane.
With the calculated significance values, the insignificant (weak) features can be discarded. The resulting distribution map for the urban traffic scene in figure 6.5 is illustrated in figure 6.6. Figure 6.6a shows the distribution map generated with all features, whereas for building the distribution map in figure 6.6b, one third of the feature classes with the lowest significance for the context class *Urban* where excluded. The excluded classes are *Car Complete, Car Left, Car Right, Error Grass Complete,* and *License Plate.*

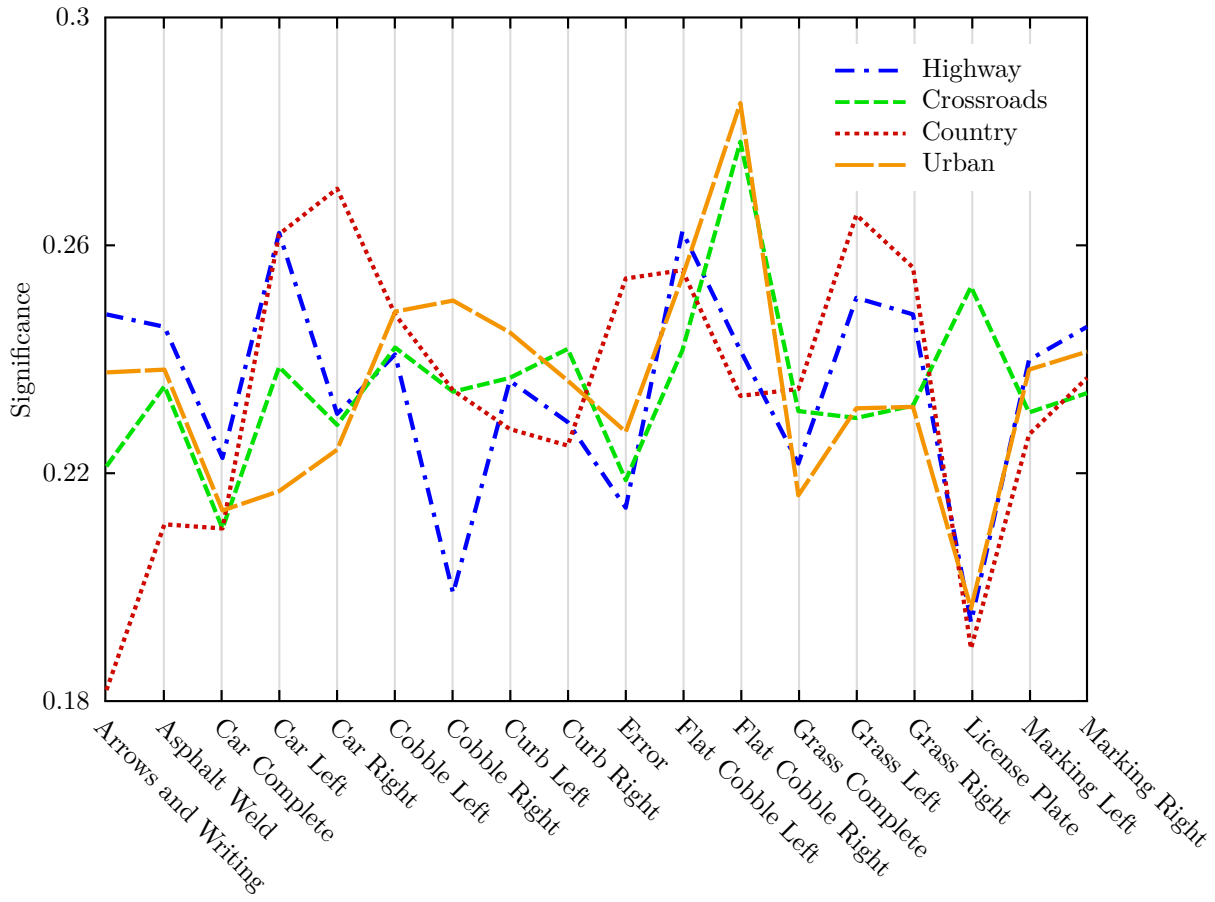


Figure 6.4.: Significance of the spatial relations between different feature classes and the vehicle trajectory

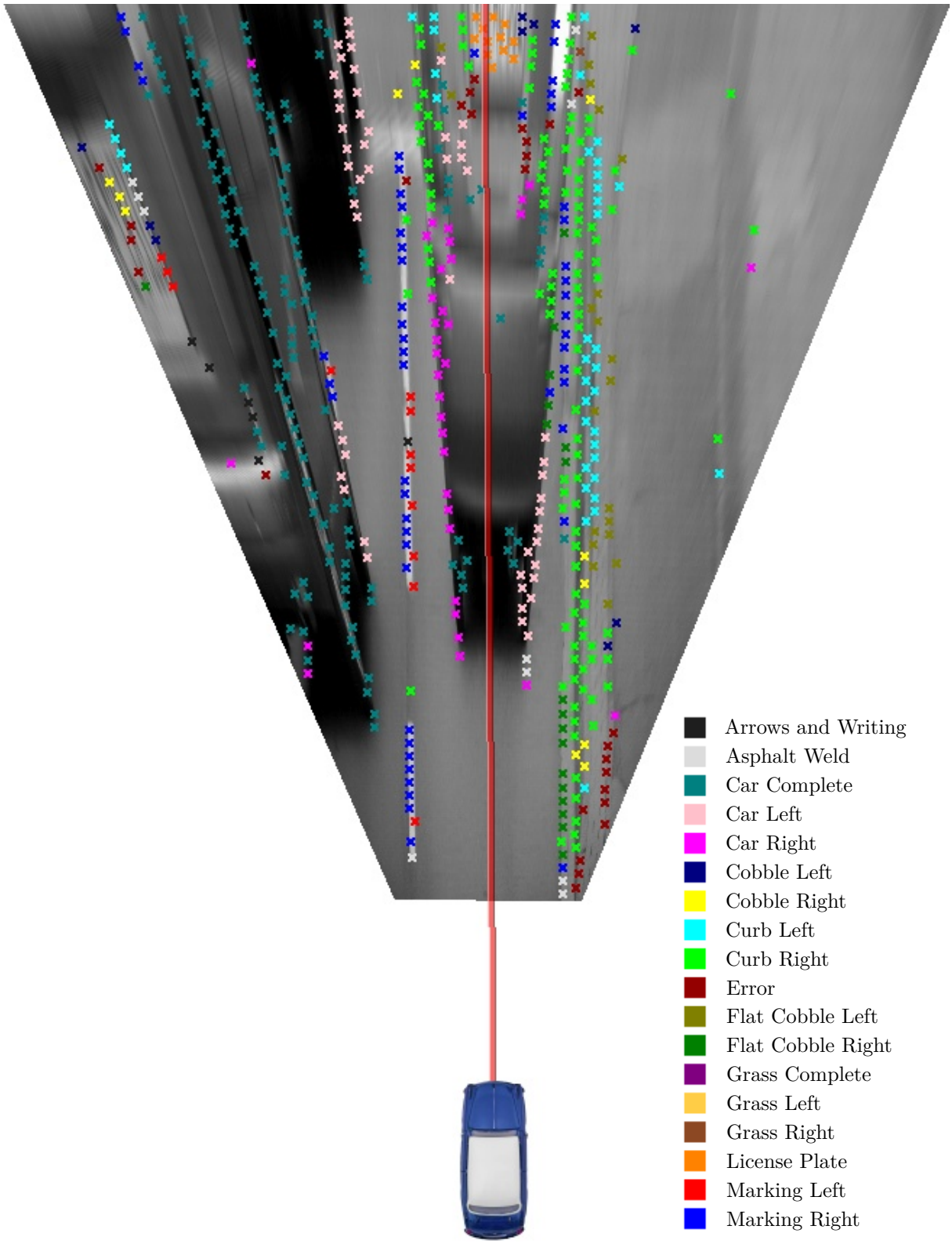
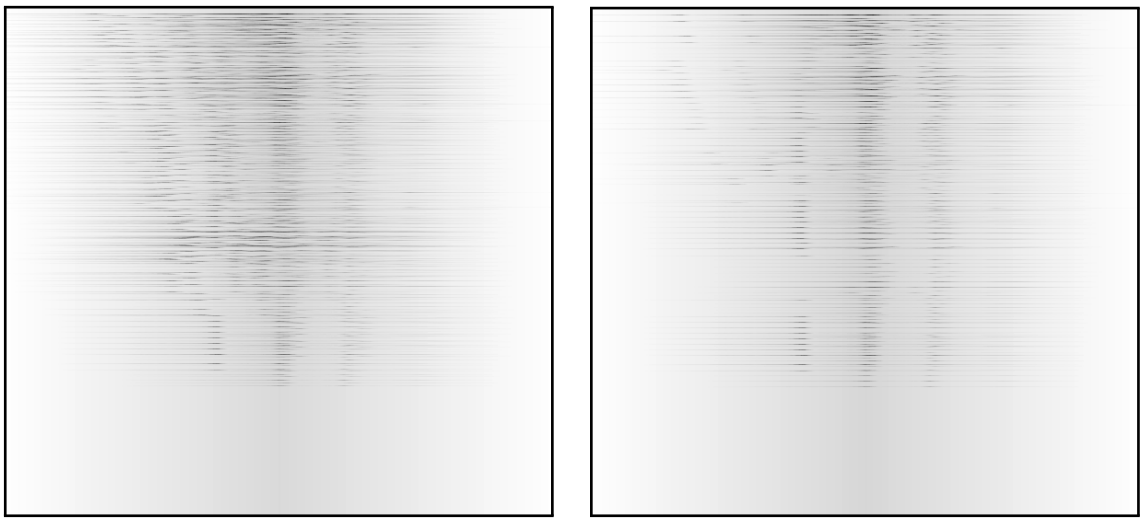


Figure 6.5.: Example urban scene with estimated trajectory (Dynamic Programming approach)



(a) All features **(b)** Some feature classes excluded

Figure 6.6.: Distribution map with all features (a) and without the features excluded by the context-based feature weighting (b)

With a training set which perfectly represents reality (and the testing set), the lateral offset histograms of features which do not provide useful information would actually appear as uniform distributions. If the histogram differs from a uniform distribution, the feature indeed provides information about the location of the trajectory, exactly as learned from the training set. However, the training set does generally not represent reality perfectly, so even for unnecessary features, the offset histograms would always pretend to have a certain amount of information. Figure 6.6 shows that the exclusion of weak features can support the lane matching process by providing more distinct peaks at the trajectory location, although the spatial relations of the excluded features have been learned correctly, according to the training data. Anyway, the main purpose of excluding a certain part of the features from the distribution map generation is to save computing time. Especially in situations like in figure 6.5, where a big region of the projected image is occupied by weak features like *Car Complete* (or, e.g., *Grass Complete* in other situations), a lot of time in the distribution map registration process can be saved by discarding these features. Furthermore, the more important features like *Curb Right* or *Marking Left* are emphasized, so in addition to the reduced computing time, the quality of the algorithm can be improved.

6.5. Experiments with other Datasets

This framework was also tested with camera images from a public dataset, the KITTI Road dataset (Fritsch et al., 2013) from the KITTI Vision Benchmark Suite (Geiger et al., 2012). Since this dataset only contains labeled road regions and driving corridors for testing road segmentation algorithms and does not provide ground truth trajectories, because it consists of single non-consecutive images, it could only be tested qualitatively if the presented approach also works with other data. The experiments also illustrate the influence of different camera characteristics. Figure 6.7 and figure 6.8 show successful example results of the presented trajectory estimation approach. For these tests, no parameters were changed, even the spatial relations, learned with the other dataset, were used. Only the maximum distance from the vehicle (and so the size of the bird's eye view image) was changed from 30 meters to 40 meters. An example of a failure is shown in figure 6.9. More example results with the KITTI dataset can be found in appendix C (figures C.1 to C.3). Some issues related to different camera characteristics can be observed. The images from the KITTI dataset show a much higher contrast and sharper edges than the dataset used for training the classifier for the local features. For this reason, the local feature classification did not work well within this dataset, as can be seen in figures 6.7 to 6.9. Of course, the classifier does not have to be trained individually for every new camera, but a high classification performance can only be achieved if the camera characteristics (resolution, field of view, contrast etc.) are similar. Furthermore, the KITTI camera has a much wider field of view, with several consequences for lane detection systems:

- Tight curves still appear in the camera image, which allows a better near range environment perception and lane detection, especially in urban and crossroad scenarios
- The pixel depth (pixels per meter) in far range decreases rapidly, with the effect that in the BEV image, the feature detection and classification performance also decreases
- The camera image contains completely new views of the vehicle environment, e.g., nearly side views of other vehicles (the field of view could be decreased by cutting off the border regions, but this would also decrease the resolution)

The last point and the poor local feature classification performance made a reliable context classification with these images impossible, because the training and the testing (KITTI) datasets differ too much. In this case, the training dataset just does not represent reality (the testing set) adequately. Therefore, the trajectory estimation in figure 6.7 to 6.9 was performed by applying the curve template matching, without context classification. However, this experiment shows that the presented approach also works with other datasets, as long as the conditions in the training and testing phase do not differ too much.

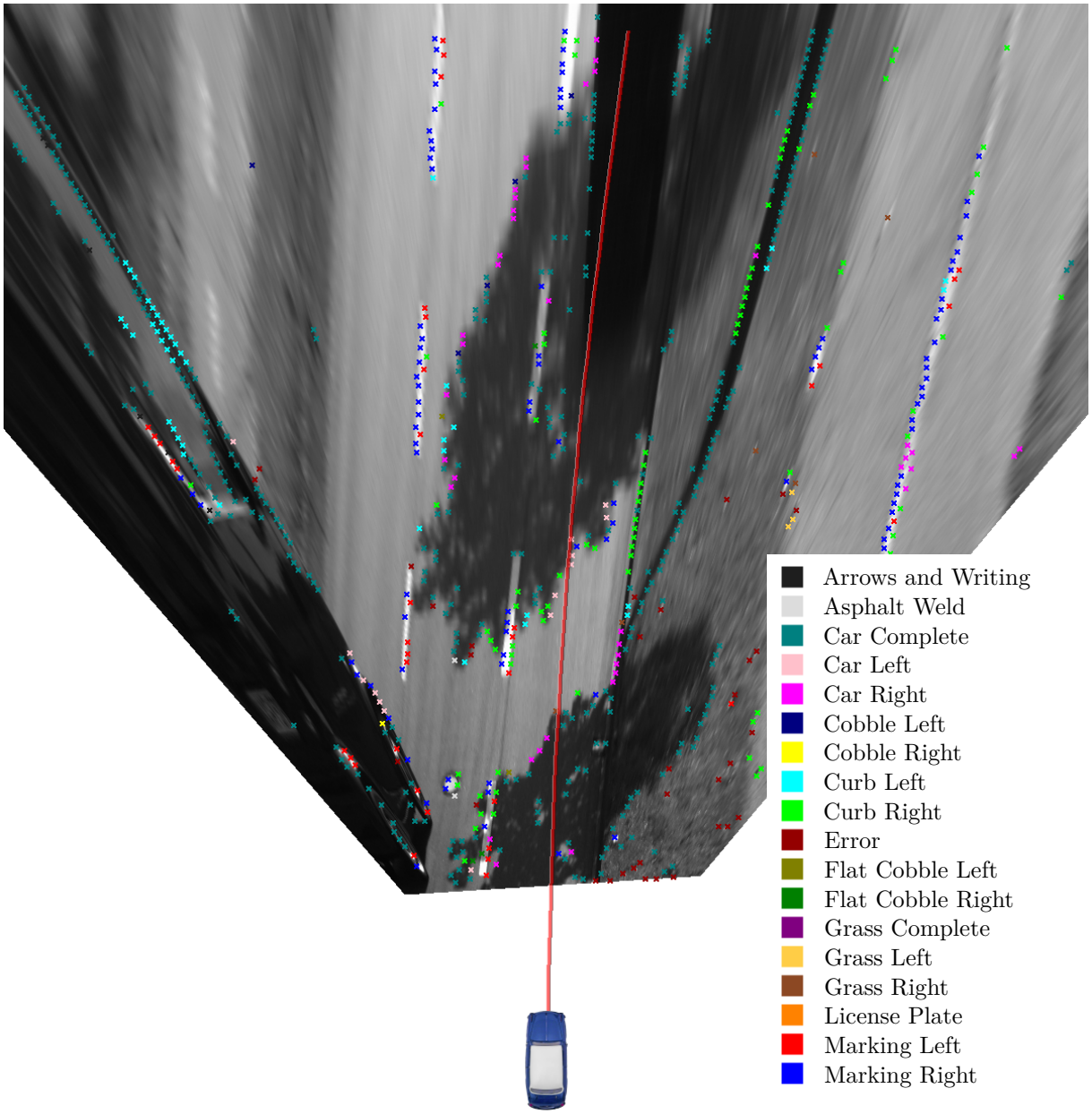


Figure 6.7.: Result with an image from the KITTI Road dataset (Fritsch et al., 2013)

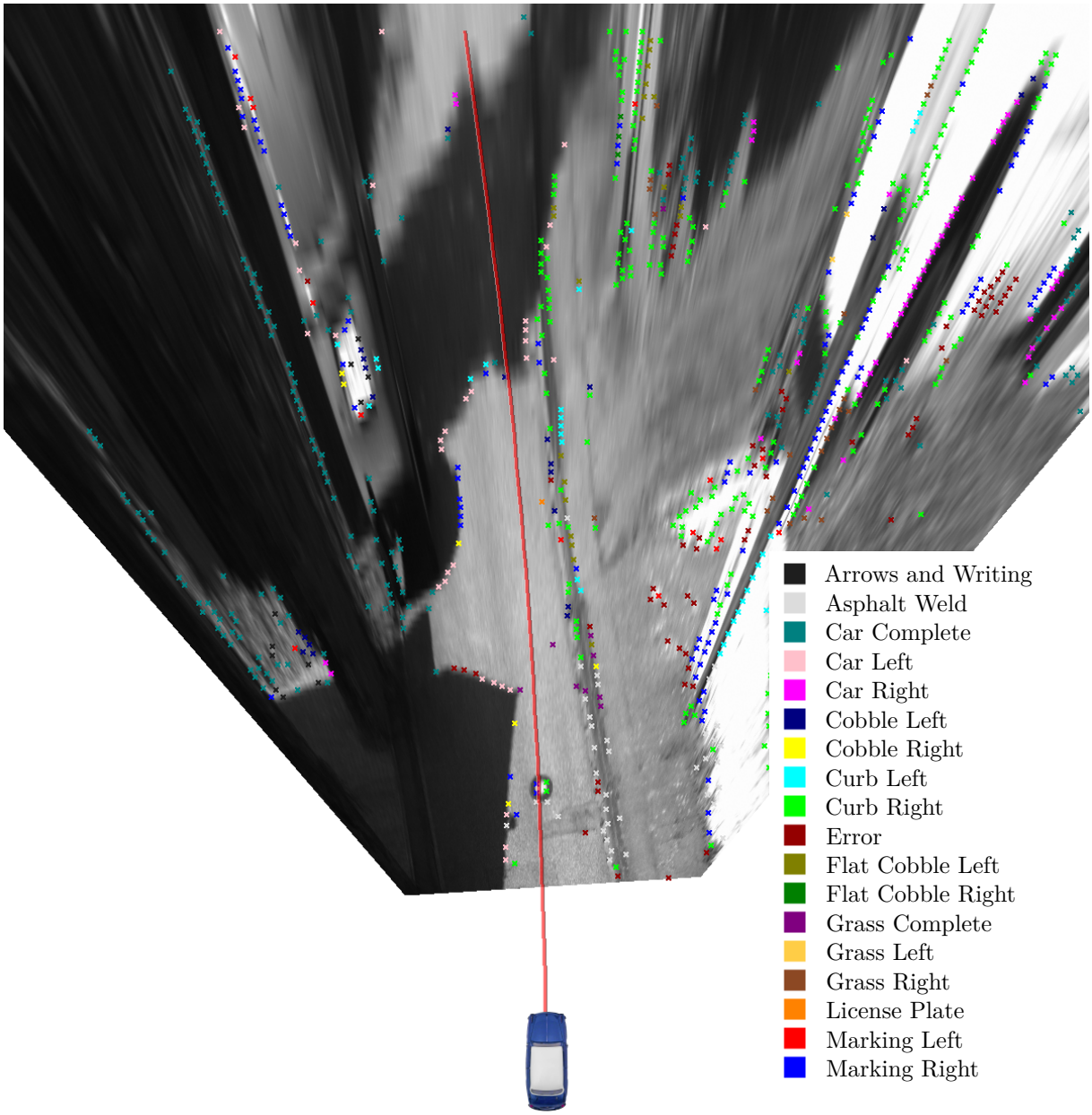


Figure 6.8.: Result with an image from the KITTI Road dataset (Fritsch et al., 2013)

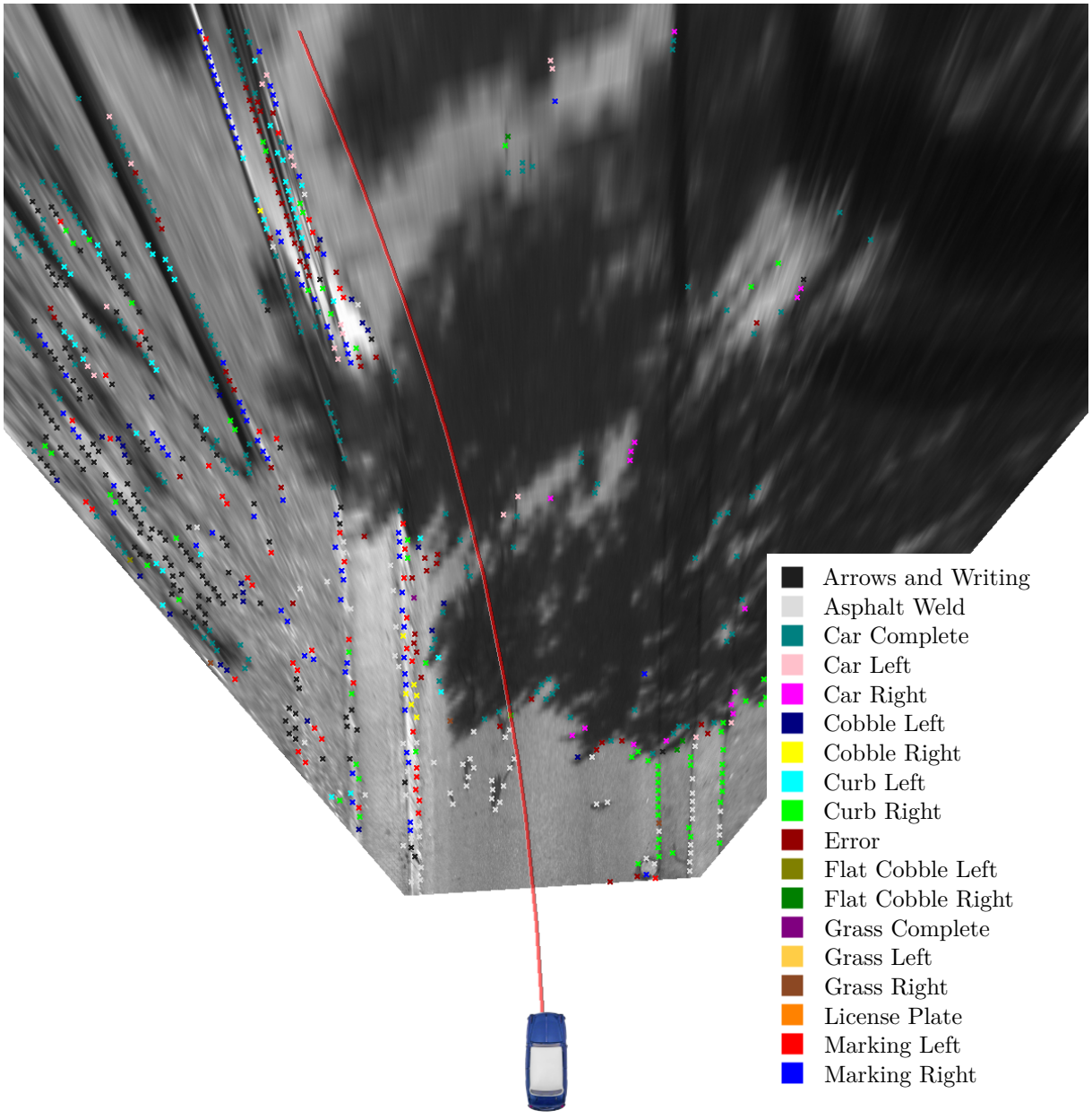
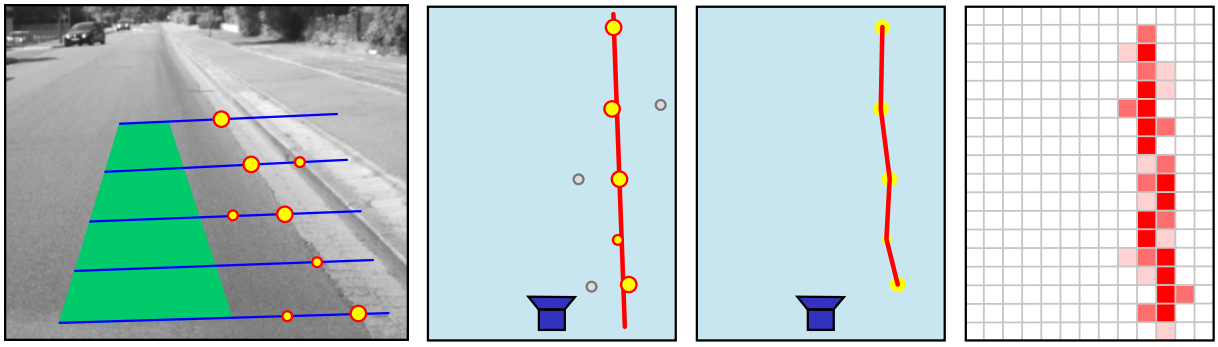


Figure 6.9.: Result (failure) with an image from the KITTI Road dataset (Fritsch et al., 2013)

6.6. Lane Border Detection

Until here, a method for estimating the vehicle trajectory or the coarse location of the lane has been presented. In real applications in the area of advanced driver assistance systems up to fully automated driving, knowledge about the exact lane course is required. There are many approaches for detecting the lane or the segmentation of the camera image into road and no road (see section 1.2). A very robust approach for camera-based lane border detection in arbitrarily structured environments (Schomerus et al., 2014) detects all types of lane borders very precisely. However, it is a local approach, which means that it needs a starting position (location of the free space in front of the car) for the lane border search. Starting at the previously defined free space, certain profile lines in the image are analyzed for strong signal changes. At locations of these signal changes, lane border candidates are added as features, and the Weighted RANSAC (see section 4.2.1) is used to detect the line which best represents the lane border. The coarse procedure is illustrated in figure 6.10 (Schomerus et al., 2014). Some results of the lane border detection are shown in figure 6.11 (Schomerus et al., 2014).



(a) (b) (c) (d)

Figure 6.10.: Lane border detection (Schomerus et al., 2014): a) Feature detection on profile lines. The green area is the defined free space. b) Line fitting with projected features, c) Polyline generation, d) Probabilistic grid map update.

Despite its robustness, the detection range of this method is limited because the certainty about the free space location decreases rapidly in far range. In this work, on the other hand, a global approach is presented which analyzes the complete projected image. No prior knowledge about the course of the lane is required (although such knowledge can be integrated, as described in section 2.3.2). The features are detected and classified in the whole BEV image, and a possible trajectory is estimated. The result of this global approach could now be used perfectly as a starting point for the lane border detection and ensure a correct search area also in far distance from the vehicle. Furthermore, the local feature classification could be used to classify the detected lane border candidates,

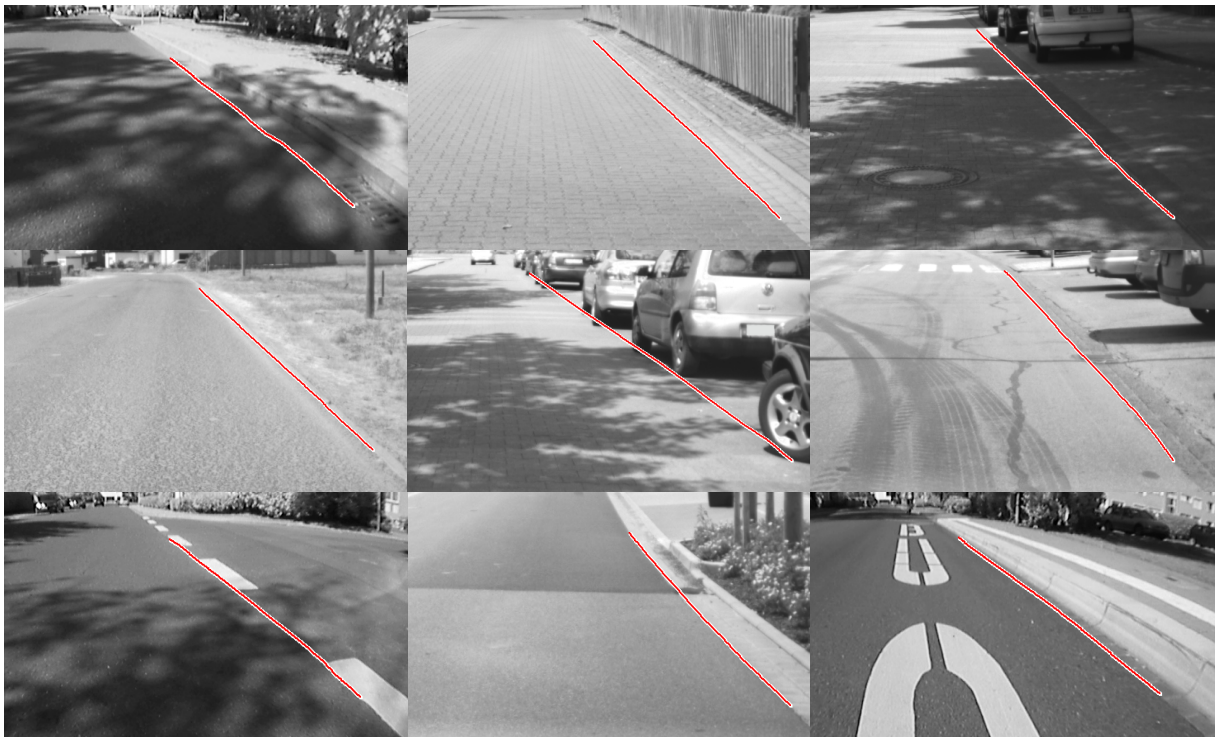


Figure 6.11.: Lane border detection results (Schomerus et al., 2014)

since a lane border consisting of equal features (curb, grass, etc.) is much more probable. Even classified context could be taken into account to predict the expected type of lane border features, e.g., a curbstone is much more probable in an urban scenario than on a highway. In this manner, the presented approach can be used as initial solution to many exact lane or lane border detection algorithms, including methods for texture-based free space detection, where the estimated trajectory specifies the location of reference texture for the free space segmentation.

Chapter 7

Conclusion and Outlook

7.1. Discussion

Although the presented method for context-supported lane detection shows very promising results, certain conditions can affect the performance in a negative way.
As could be seen in figure 6.2, for many situations, the narrow field of view of the camera used for obtaining the dataset is not sufficient. Important areas may not even appear in the camera image, and especially in the near range, close to the vehicle, many features are not visible which could provide important information about the course of the lane. Furthermore, the maximum distance of reliably classifiable features is limited by the resolution of the camera. While a camera with a wider field of view (e.g., figure 6.7) provides more information in the near range, the resolution in the far range decreases drastically which constraints the detection and classification of features. To overcome this problem, either a camera with a wide field of view and a very high resolution should be used, or two or more cameras with different fields of view could be combined.
Anyway, without depth information, the locations of features above the ground plane (e.g. vehicles, grass, etc.) are not determined correctly, due to the mapping of the camera image onto the ground plane (section 2.2.2). This leads to incorrect spatial relations. If the distance of the feature from the vehicle would be obtained by, e.g., structure from motion, stereo cameras, laser or other sensors, or methods providing depth information, the spatial relations of these features could be learned correctly.
As discussed in section 6.1, the dataset contains turns and lane changes which were not excluded for training and testing. While in the training phase, the false spatial relations due to lane changes can be compensated by the much greater number of frames with correct vehicle behavior, these unexpected driving maneuvers cause noticeable errors in the evaluation. However, the motivation of not cleaning the dataset manually from these maneuvers was to examine the method's qualification for dealing with much bigger datasets,

where such a preprocessing step would not be feasible. Since the usage of the unprocessed virtual ground truth data was successful, much more of this type of data could be used, e.g., in the context of Deep Learning, for training complex neural networks with several layers and a large number of parameters (Bengio, 2009; LeCun et al., 2015; Schmidhuber, 2015). This could allow a vehicle trajectory estimation directly from camera images as input for the deep networks.

There are many other ways to improve the performance of the proposed method. Considering the appearance of the distribution maps (see, e.g., figure 6.6b), it is obvious that also other lanes could be extracted from the learned spatial relations. The curve template matching (section 4.2.3) already detects several curves in the distribution map (figure 4.9) to increase the certainty about the lane. By applying certain constraints about the relative locations of the detected curves, the matching could be modified to detect neighboring lanes, too. This could be even improved by incorporating information from a map, including the number of lanes at the current location of the vehicle.
Another way of improving the performance is to apply a temporal tracking. Note that no temporal information is used in the evaluation of the method, the result is always determined only from a single frame. Temporal tracking usually improves the results because sporadic errors can be compensated. On the other hand, a well parameterized tracking can make a method appear better than it is, so the real characteristics may not be visible. For this reason, no temporal tracking was used for the evaluation in this work. However, a simple way to take advantage of previous results is to update the distribution map according to a probabilistic grid map, instead of creating a new distribution map for every frame. The pixels of the distribution map would correspond to grid cells and their values could be calculated by applying a probabilistic update rule (Moravec and Elfes, 1985), so the value of each cell does not only depend on the spatial relations of features in the current frame, but also on the values from previous time steps.
Furthermore, the classification of local features could be enhanced by using other feature descriptors which are more invariant against different orientations. Adding invariance usually means throwing away information which can lead to decreased performance in cases where this invariance is not necessary. But for the training of the local feature classifier (section 3.1.2), almost all used feature samples have vertical edges which leads to problems in scenarios with strong curves. So, the use of orientation invariant feature descriptors could be helpful, especially if a camera with a wider field of view is used which allows stronger curves to be visible in the image.

7.2. Conclusion

By determining the context of the scene and the semantic meaning of local features, regarding their location relative to the vehicle trajectory, this work is an important step

from image segmentation and object detection towards comprehensive scene understanding.
Using self-made ground truth data without the need of manual data labeling (virtual ground truth), the presented approach enables the application of powerful machine learning methods requiring large training datasets, to the task of image-based lane detection. For the classification of local features, a data base was created containing more than 3500 features, manually assigned to 18 feature classes which typically appear in traffic scenarios. For these local features, the spatial relations to the vehicle trajectory were learned. The registration of the spatial relations for all local features of one video frame leads to a distribution map which allows the matching of a lane model. The result of this matching can be interpreted as the most probable vehicle trajectory or the center of the lane, and thus can be used as an initial solution for an exact lane detection or a starting point for a lane border search.
Several possibilities to improve the lane detection performance with additional information about the current scene context were analyzed. For this purpose, four context classes were defined and a classifier was trained with manually labeled image frames. The presented method analyzes the complete image and does not concentrate on only certain regions of interest. It was proposed how to combine this global approach with a local method for lane border detection to improve the performance and to extend the range of coverage.
Although chapter 6 proposes to select the best model matching method depending on the context, it is also possible to perform the model matching with all matching methods and thus obtain a reliability measure by comparing the different solutions. Each module of the approach can be replaced easily by another method for feature detection, classification or model matching. Also new feature and context classes can be added for further improvement. This just requires a retraining of the corresponding classifier, of course. Furthermore, the underlying concept of this approach permits the integration of more types of sensors with different fields of view. Depth sensors, e.g., radar or laser, can be integrated for obstacle detection, and observed features alongside and behind the vehicle can also be integrated into the distribution map since they also provide information about the vehicle trajectory. For automatic driving in complex scenarios, the observation of the complete vehicle surrounding can be very helpful. Thus, algorithms are needed which can process the information of this complete environment.

While automatic driving in well-structured highway scenarios with lane markings is already performed by modern advanced driver assistance systems, in the case of automatic driving in arbitrary scanarios, the complexity of urban traffic scenes make great demands on sensor setups and algorithms for environment perception and vehicle control. By estimating the scene context and the semantic meaning of local features, this work is a contribution to tread the path from image segmentation and object detection towards comprehensive scene understanding, which is the prerequisite for technical solutions for safe and comfortable mobility in the future.

Appendix A

Model Matching Results

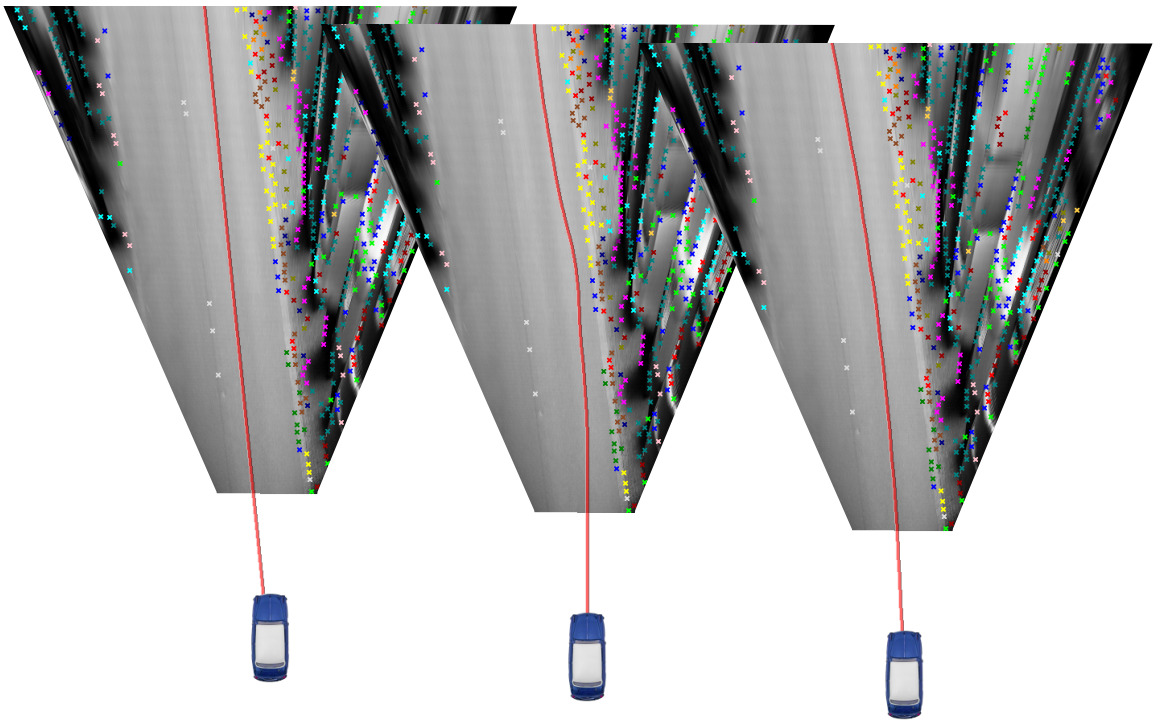


Figure A.1.: Trajectory estimation with different models: random line search (left), dynamic programming (center), curve template matching (right)

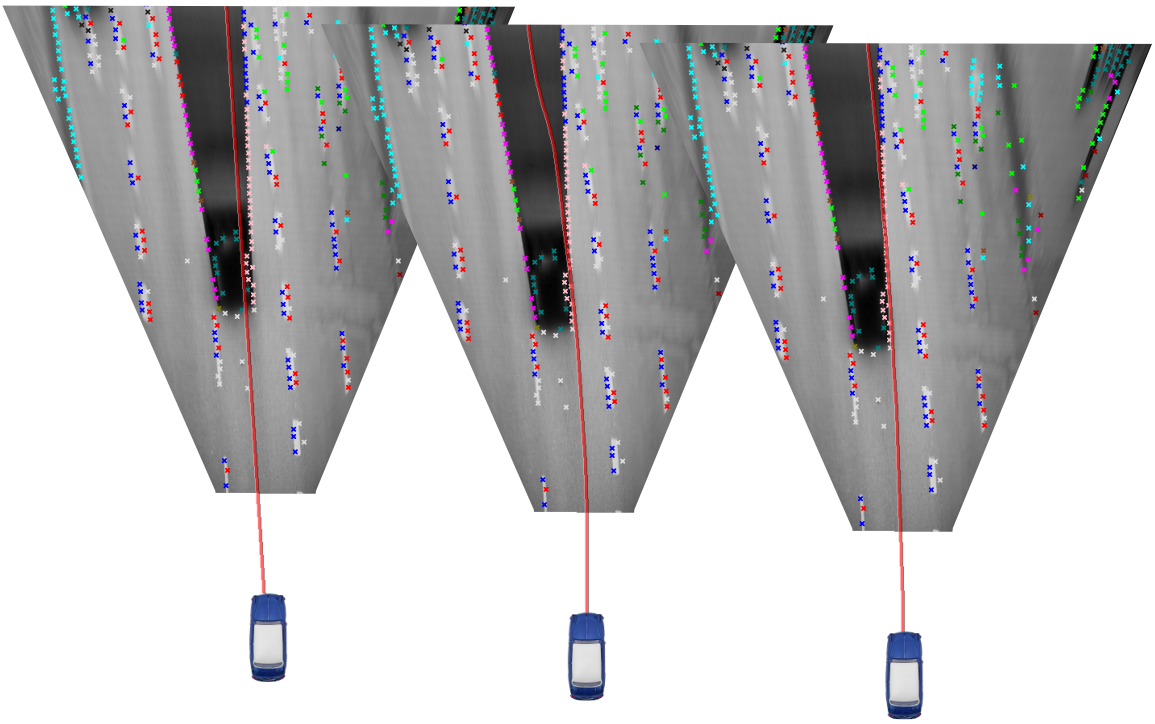


Figure A.2.: Trajectory estimation with different models

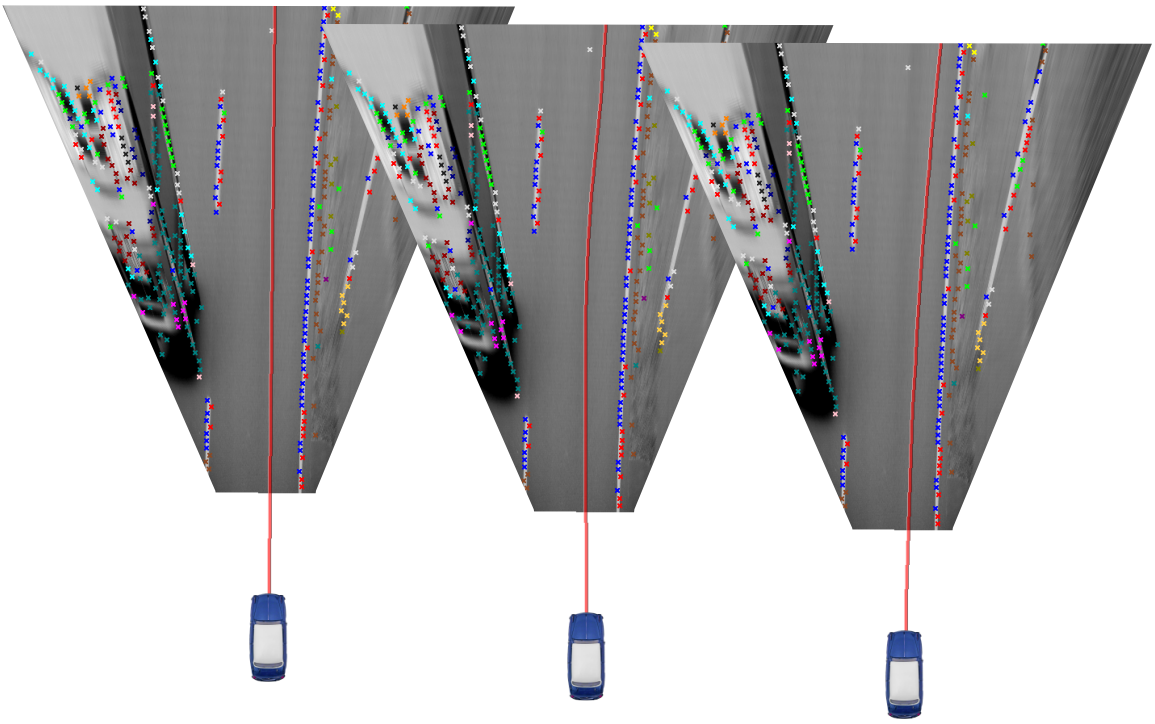


Figure A.3.: Trajectory estimation with different models

Appendix B

Context Classification Results

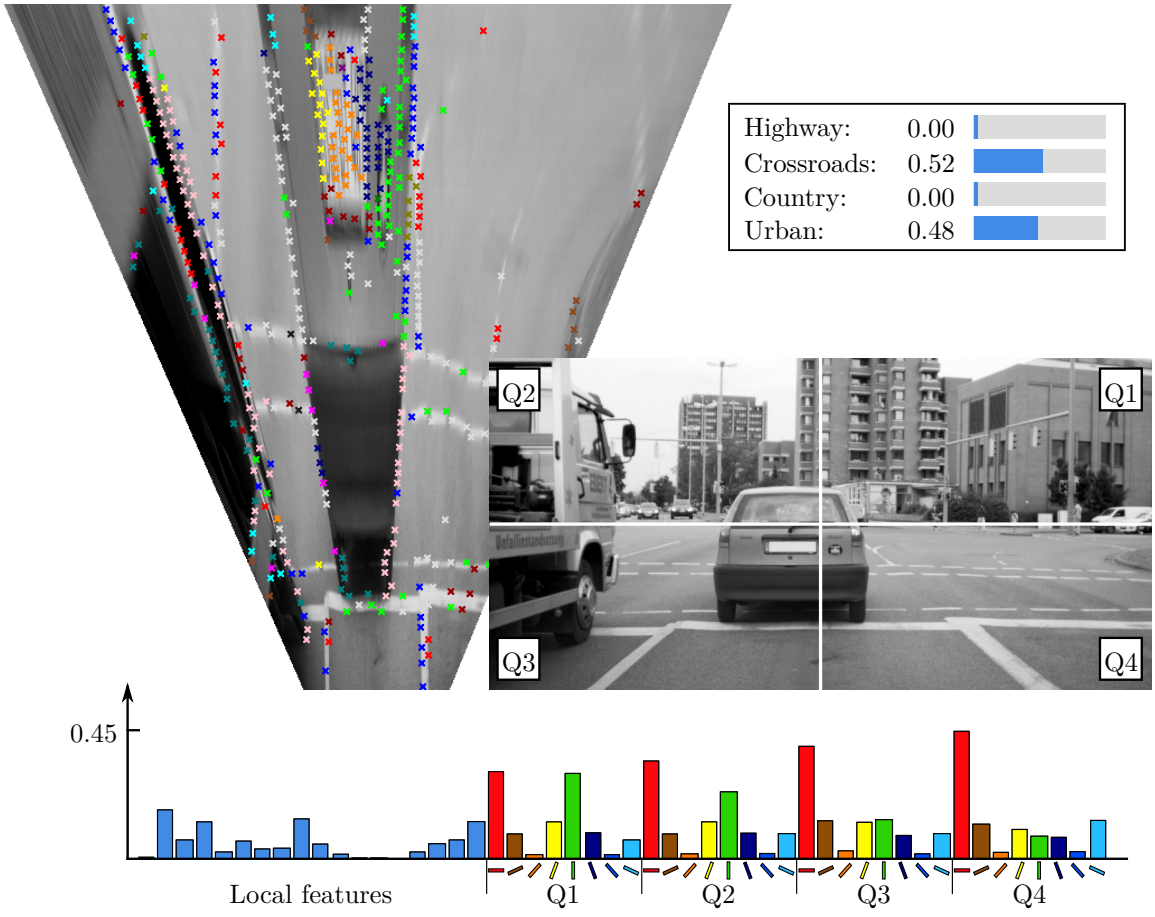


Figure B.1.: Context classification result: camera image with subregions Q1-Q4, BEV image with local features, and the global feature vector

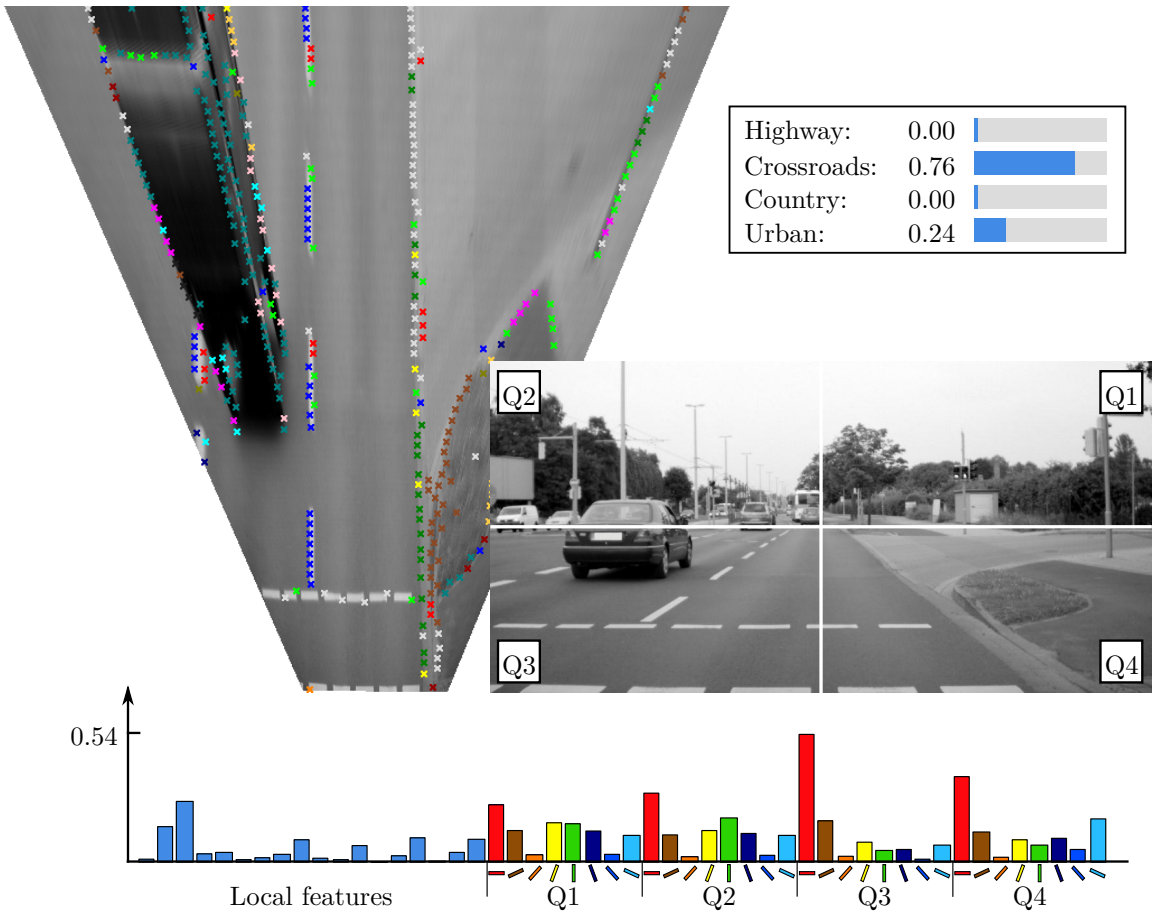


Figure B.2.: Context classification result: camera image with subregions Q1-Q4, BEV image with local features, and the global feature vector

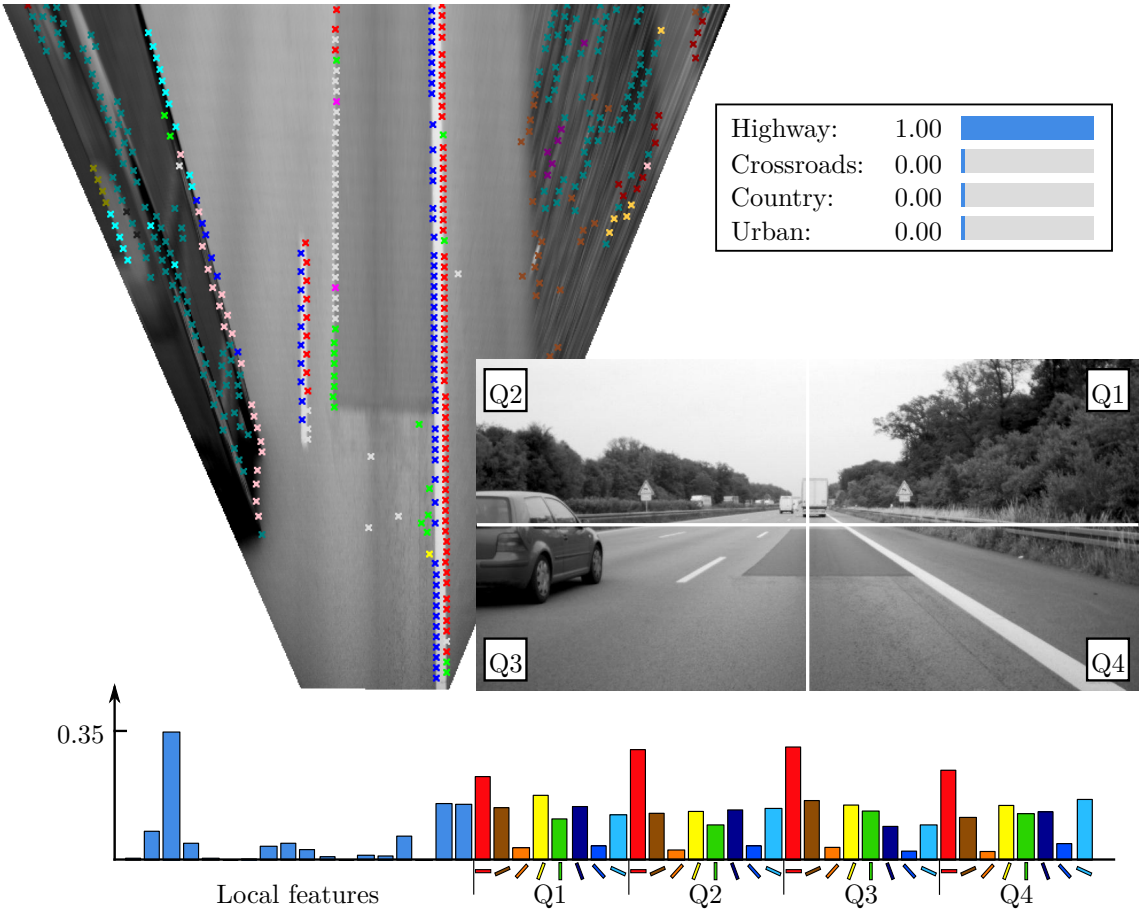


Figure B.3.: Context classification result: camera image with subregions Q1-Q4, BEV image with local features, and the global feature vector

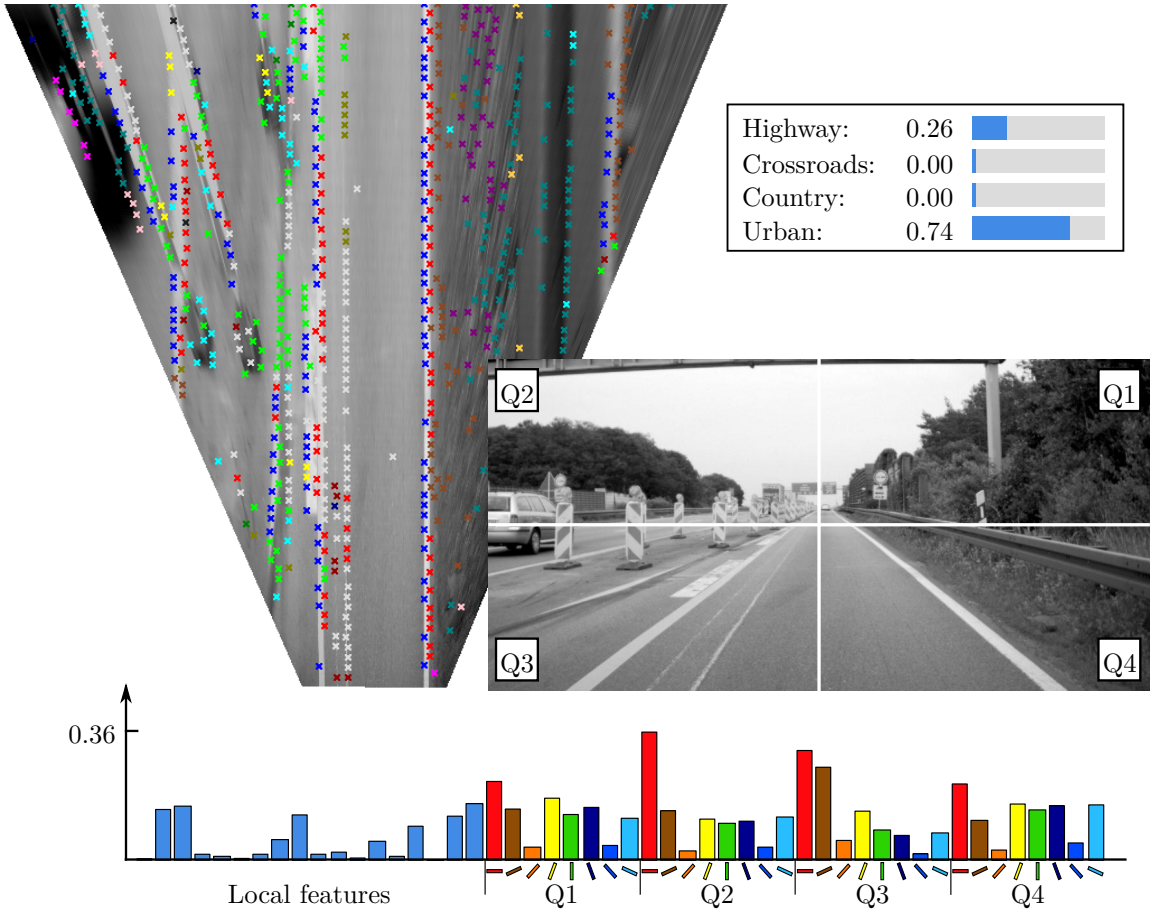


Figure B.4.: Context classification result (failure): camera image with subregions Q1-Q4, BEV image with local features, and the global feature vector

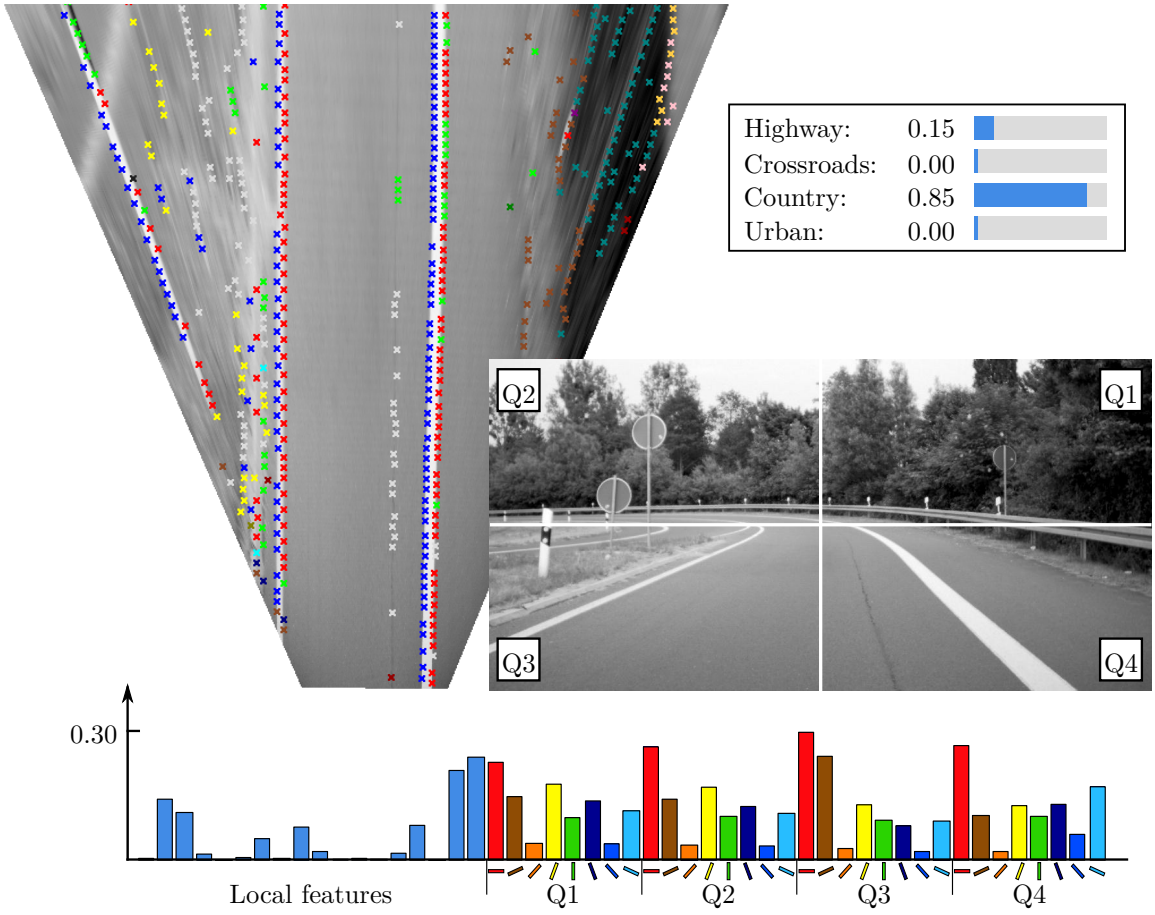


Figure B.5.: Context classification result (failure): camera image with subregions Q1-Q4, BEV image with local features, and the global feature vector

Appendix C

Results with images from the KITTI dataset

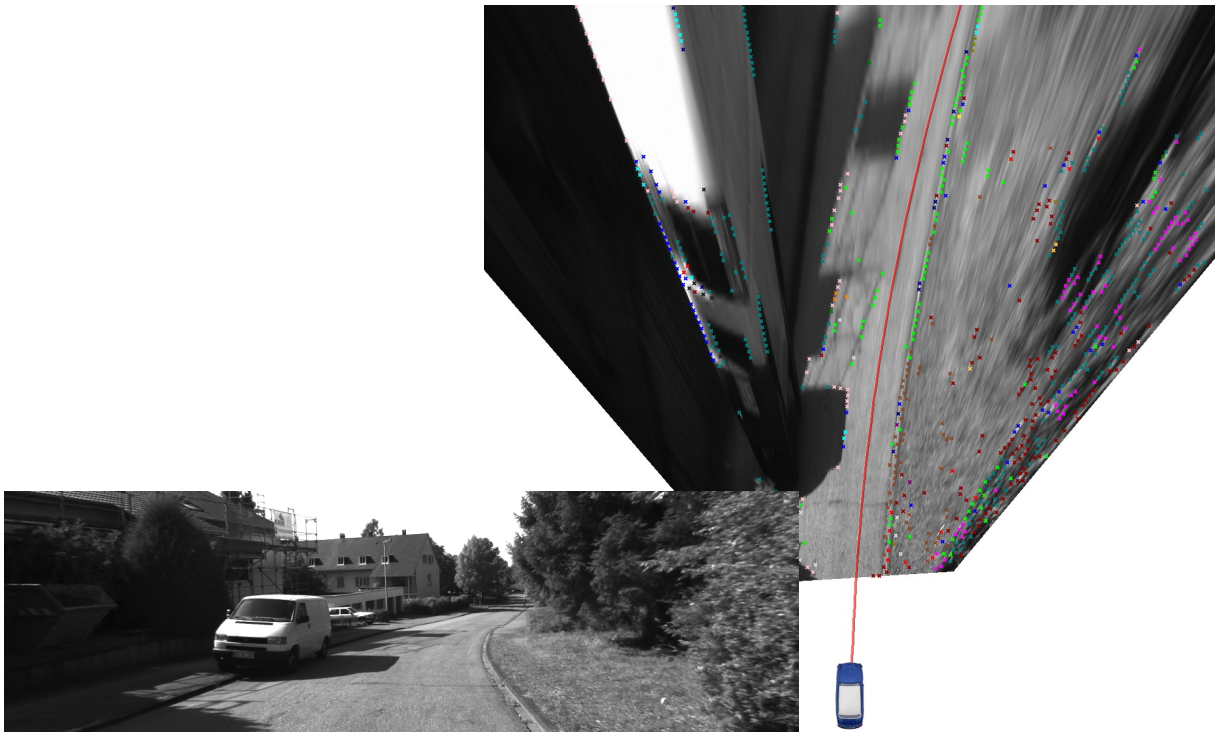


Figure C.1.: Result of the lane detection with the KITTI dataset

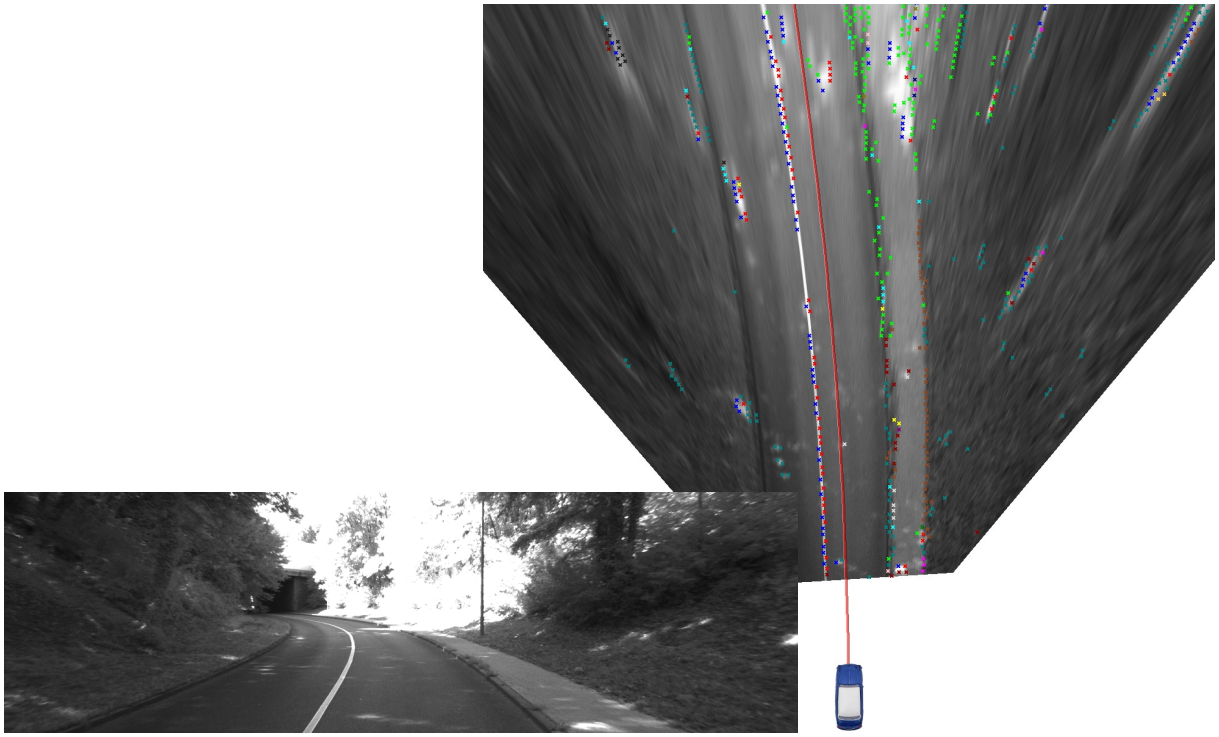


Figure C.2.: Result of the lane detection with the KITTI dataset

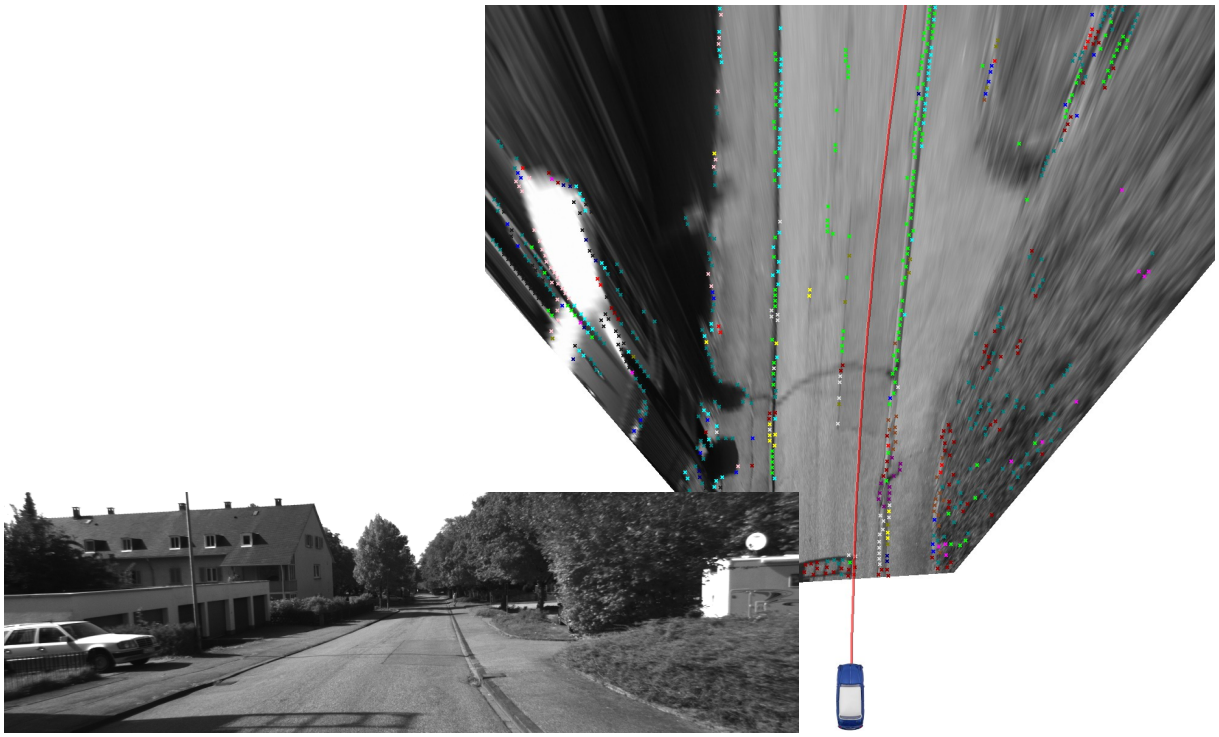


Figure C.3.: Result of the lane detection with the KITTI dataset

Notation

$\mathbf{A}$	Matrix
$\mathbf{A}^\top$	Transposed matrix
$\mathbf{A}^{-1}$	Inverse matrix
${}^A\mathbf{T}_B$	Homogeneous transformation matrix between coordinate systems A and B
m	2D point (2D vector)
$\tilde{m}$	2D point in homogeneous coordinates (3D vector)
M	3D point (3D vector)
$\tilde{M}$	3D point in homogeneous coordinates (4D vector)
$\Psi(x, y)$	The distribution map
$\xi(x, y)$	The attractiveness map
$h_\beta(x)$	Hypothesis function for sample x
β	Regression parameters
$J(\beta)$	Cost function with parameters β
$\tilde{\tau}$	Ground truth trajectory
ζ	Significance of lateral offset histogram

Glossary

Bird's Eye View	Image projected onto the ground plane
Camera Calibration	Determination of camera parameters
Canny Edge Detection	Popular algorithm for edge detection in gray value images
Global Feature	Entity of information without a certain location in the image
GPS	Global Positioning System
IMU	Inertial measurement unit
LiDAR	Time-of-flight laser sensor for distance measurements
Local Feature	Entity of information assigned to a certain location in the image
Occupancy Grid Map	A map dividing the considered area into cells of certain size, which are either free or occupied by an obstacle
Probabilistic Grid Map	An Occupancy Grid Map, which assigns occupancy probabilities to the cells allowing a temporal filtering
Signal-to-noise Ratio (SNR)	Measure for the quality of a signal (e.g., an image), affected by noise
Sobel Filter	Filter for simultaneous image smoothing and edge detection

List of Figures

Bibliography

Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. In *Machine Learning*, pages 37–66.

Alahi, A., Ortiz, R., and Vandergheynst, P. (2012). FREAK: Fast Retina Keypoint. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 510–517.

Alon, Y., Ferencz, A., and Shashua, A. (2006). Off-road Path Following using Region Classification and Geometric Projection Constraints. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 689–696.

Alvarez, J. M., Gevers, T., and Lopez, A. M. (2010). 3D Scene priors for road detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 57–64. Ieee.

Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515.

Bengio, Y. (2009). Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127.

Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6).

Chen, T., Dai, B., Liu, D., Song, J., and Liu, Z. (2015). Velodyne-based Curb Detection Up to 50 Meters Away. In *IEEE Intelligent Vehicles Symposium*, pages 241 – 248.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 886–893.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38.

Duda, R. O. and Hart, P. E. (1972). Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15.

Faktor, A. and Irani, M. (2014). "Clustering by Composition"—Unsupervised Discovery of Image Categories. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 36, pages 1092–1106.

Felisa, M. and Zani, P. (2010). Robust monocular lane detection in urban environments. In *IEEE Intelligent Vehicles Symposium*, pages 591–596.

Fischler, M. A. and Bolles, R. C. (1981). Random Sample Consensus: A Paradigm for Model Fitting with Applicatlons to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381 – 395.

Frank, E. and Witten, I. H. (1998). Generating accurate rule sets without global optimization. In Shavlik, J., editor, *International Conference on Machine Learning*, pages 144–151. Morgan Kaufmann.

Fritsch, J., Kuehnl, T., and Geiger, A. (2013). A new performance measure and evaluation benchmark for road detection algorithms. In *IEEE International Conference on Intelligent Transportation Systems*.

Gao, Q., Luo, Q., and Moli, S. (2007). Rough Set based Unstructured Road Detection through Feature Learning. In *IEEE International Conference on Automation and Logistics*, pages 101–106.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Haralick, R. M., Shanmugam, K., and Dinstein, I. (1973). Textural Features for Image Classification. In *IEEE Transactions on Systems, Man and Cypernetics*, pages 610–621.

Hoiem, D., Efros, A. a., and Hebert, M. (2008). Putting Objects in Perspective. *International Journal of Computer Vision*, 80(1):3–15.

Huang, A. S., Moore, D., Olson, E., and Teller, S. (2008). Multi-Sensor Lane Finding in Urban Road Networks. In *Proceedings of Robotics: Science and Systems*.

Kammel, S. and Pitzer, B. (2008). Lidar-based lane marker detection and mapping. In *IEEE Intelligent Vehicles Symposium*, pages 1137–1142.

Kluge, K. (1994). Extracting road curvature and orientation from image edge points without perceptual grouping into features. In *IEEE Intelligent Vehicles Symposium*, pages 109–114. Ieee.

Kong, H., Audibert, J.-Y., and Ponce, J. (2010). General road detection from a single image. *IEEE Transactions on Image Processing*, 19(8):2211–2220.

Kuhnl, T., Kummert, F., and Fritsch, J. (2011). Monocular road segmentation using slow feature analysis. In *IEEE Intelligent Vehicles Symposium*, pages 800–806. Research Institute for Cognition and Robotics, Bielefeld University, Germany.

Landwehr, N., Hall, M., and Frank, E. (2005). Logistic model trees. *Machine Learning*, 59(1-2):161–205.

Lazebnik, S., Schmid, C., and Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 2169–2178.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Lim, K. H., Ang, L.-m., Seng, K. P., and Chin, S. W. (2009). Lane-Vehicle Detection and Tracking. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume II, pages 5–10.

Linarth, A. and Angelopoulou, E. (2011). On feature templates for Particle Filter based lane detection. In *IEEE International Conference on Intelligent Transportation Systems*, pages 1721–1726.

Loose, H. and Franke, U. (2010). B-spline-based road model for 3d lane recognition. In *IEEE International Conference on Intelligent Transportation Systems*, pages 91–98.

Mallot, H. A., Bülthoff, H. H., Little, J. J., and Bohrer, S. (1991). Inverse perspective mapping simplifies optical flow computation and obstacle detection. *Biological Cybernetics*, 64(3):177–185.

Mark Hall, Eibe Frank, G. H. B. P. P. R. I. H. W. (2009). The weka data mining software: An update. *SIGKDD Explorations*, 11(1).

Milgram, J., Cheriet, M., and Sabourin, R. (2006). "One Against One" or "One Against All": Which One is Better for Handwriting Recognition with SVMs? In *Tenth International Workshop on Frontiers in Handwriting Recognition*, pages 1–6.

Moravec, H. P. and Elfes, A. (1985). High resolution maps from wide angle sonar. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 116–121.

Ojala, T., Pietikäinen, M., and Harwood, D. (1994). Performance Evaluation of Texture Measures with Classification Based on Kullback Discrimination of Distributions. In *IEEE International Conference on Pattern Recognition*, volume 1, pages 582–585.

Oliva, A. and Torralba, A. (2001). Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175.

Paetzold, F. and Franke, U. (1998). Road Recognition in Urban Environment. In *IEEE International Conference on Intelligent Vehicles*, pages 87–91.

Parzen, E. (1962). On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076.

Pelaez, G. A., Bacara, D., de la Escalera, A., Garcia, F., and Olaverri-Monreal, C. (2015). Road Detection with Thermal Cameras through 3D Information. In *IEEE Intelligent Vehicles Symposium*, pages 255–260.

Perlich, C., Provost, F., and Simonoff, J. S. (2003). Tree Induction vs. Logistic Regression: A Learning-Curve Analysis. *Journal of Machine Learning Research*, 4:211–255.

Pomerleau, D. (1995). RALPH: rapidly adapting lateral position handler. In *IEEE Intelligent Vehicles Symposium*, pages 506–511.

Quinlan, R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA.

Renninger, L. W. and Malik, J. (2004). Probability Estimates for Multi-class Classification by Pairwise Coupling. *Journal of Machine Learning Research*, 44(19):2301–2311.

Rosten, E. and Drummond, T. (2005). Fusing points and lines for high-performance tracking. In *International Conference on Computer Vision*.

Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. *Neural Networks*, 61:85–117.

Schomerus, V., Rosebrock, D., and Wahl, F. M. (2014). Camera-based Lane Border Detection in Arbitrarily Structured Environments. In *IEEE Intelligent Vehicles Symposium*, pages 56–63.

Seibert, A., Michael, H., and Tewes, A. (2013). Camera based detection and classification of soft shoulders , curbs and guardrails. In *IEEE Intelligent Vehicles Symposium*, pages 853–858.

Shang, E., An, X., Li, J., Ye, L., and He, H. (2013). Robust Unstructured Road Detection: The Importance of Contextual Information. *International Journal of Advanced Robotic Systems*, page 1.

Sheather, S. J. and Jones, M. C. (1991). A Reliable Data-Based Bandwidth Selection Method for Kernel Density Estimation. *Journal of the Royal Statistical Society*, 53(3):683–690.

Tsai, R. Y. (1986). An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 364–374.

Wang, Y., Teoh, E. K., and Shen, D. (2004). Lane detection and tracking using B-Snake. *Image and Vision Computing*, 22(4):269–280.

Zhang, Z. (1999). Flexible camera calibration by viewing a plane from unknown orientations. In *IEEE International Conference on Computer Vision*, volume 1.

Index

R

S

U

V

W